VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Návrh databázové aplikace pro malou firmu zabývající se řemeslnou výrobou

Database Application Design for Small Business Engaged in Handicraft

Student: Václav Homola
Vedoucí bakalářské práce: Ing. Vitězslav Novák, Ph.D.

Ostrava 2013

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání bakalářské práce

Student: **Václav Homola**

Studijní program: B6209 Systémové inženýrství a informatika

Studijní obor: 6209R001 Aplikovaná informatika

Téma: Návrh databázové aplikace pro malou firmu zabývající se řemeslnou výrobou
Database Application Design for Small Business Engaged in Handicraft

Zásady pro vypracování:

1. Úvod
2. Teoretická východiska práce
3. Analýza současného stavu ve firmě
4. Realizace a implementace databázové aplikace
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků bakalářské práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

LONEY, Kevin a Marlene THERIAULT. *Mistrovství v Oracle.* Praha: Computer Press, 2002. ISBN 80-7226-635-7.
ŠIMONOVÁ, Stanislava a Jan PANUŠ. *Databázové systémy I.* Pardubice: Univerzita Pardubice, 2007. ISBN 978-80-7194-988-6.
URMAN, S., R. HARDMAN a M. MCLAUGHLIN. *Oracle: Programování v PL/SQL.* Brno: Computer Press, 2007. ISBN 978-80-251-1870-2.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 23.11.2012
Datum odevzdání: 10.05.2013

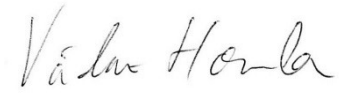
VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA Ekonomická fakulta OSTRAVA

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Místopřísežné prohlášení o samostatném vypracování bakalářské práce
„Prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně“.

Zároveň bych chtěl na tomto místě poděkovat svému vedoucímu bakalářské práce, Ing. Vítězslavu Novákovi, Ph.D., za odborné vedení, cenné rady a připomínky, které mi pomohly k vypracování této bakalářské práce.



……

Václav Homola

Datum odevzdání bakalářské práce: 10. května 2013

Obsah

1 Úvod

Hlavním důvodem zvolení tématu návrhu databázové aplikace je aktuálnost tématu a zvolených postupů v práci a autorův zájem využít své znalosti v programovacím jazyce Java a tvorbě databází pomocí jazyka SQL.

Cílem bakalářské práce je navrhnout a realizovat databázi, která bude sloužit k vedení nabídkového listu produktů, objednávek a fakturací firmy Medové pečivo zabývající se výrobou dekorativního perníku. Dalším cílem je vytvořit k databázi okenní aplikaci, díky které bude možné jednoduše a efektivně manipulovat s daty, zadávat objednávky do databáze a těm následně, pokud bude třeba, vystavit fakturu a ty vytisknout.

Práce je rozdělena do tří částí. V první, teoretické části jsou popsány základní vlastnosti databáze, objekty databáze, databázové modely a základní knihovna uživatelských prvků Javy sloužící pro vývoj desktopových aplikací. Kapitola slouží jako teoretická příprava k části praktické. Druhá část stručně shrnuje aktuální stav vedení záznamů o objednávkách a fakturách ve firmě a třetí popisuje konkrétní postup při vývoji aplikace od tvorby datové základny, navázání spojení s databázovým systémem až po samotnou okenní aplikaci. Dále jsou v této kapitole popsány základní funkce oken a jejich princip fungování je demonstrován na konkrétním kódu.

K vytvoření databáze je zvolen DBMS Oracle, jenž se v současnosti těší velké oblibě jak v malých firmách, tak velkých korporacích. Navíc poskytuje volně dostupnou databázi s omezeným, ovšem pro potřeby práce dostačujícím místem, tudíž není třeba zakupovat žádné licence. Samotná databáze bude navržena pomocí jazyka SQL a jeho procedurální nadstavby PL/SQL poskytující firmou Oracle, ve vývojovém prostředí SQL Developer. Uživatelské rozhraní bude vyvíjeno v programovacím jazyce Java pomocí programu NetBeans IDE, protože patří mezi nejrozšířenější programovací jazyky, podporuje objektově orientované programování a lze pomocí něj řešit prakticky jakýkoli problém. Java kód bude doplněn o SQL příkazy sloužící k manipulaci s daty.

2 Teoretická východiska práce

2.1 Co je databáze

Pod pojmem databáze si lze představit prakticky cokoli od jediné jednoduché tabulky, jako například telefonní seznam, adresář klientů firmy, až po složitý soubor tabulek a nástrojů tvořících systém, modelující vztahy v reálném světě. Pod tímto systémem si lze představit databázi knih v knihovně, hierarchické vztahy zaměstnanců v určité firmě a další. Důvodem zakládání databází je jednoduché a efektivní vyhledávání v datech a následná manipulace s vybranými daty. (Šimonová a Panuš, 2007).

V minulosti se používalo ke shromažďování dat zejména agendové zpracování. Agendové zpracování znamená přechod od manuálního zpracování dat k mechanizovanému a později automatizovanému zpracování. Jeho principem bylo vedení malých, na sobě nezávislých úloh, agend. Každá takováto agenda měla svoje vlastní soubory, čímž docházelo k vícenásobnému ukládání a následnému zpracování značného počtu stejných dat. Tím vznikaly následující potíže, ke kterým tímto postupem docházelo.

- **Redundance dat**. Ve dvou různých agendách se mohly nacházet duplicitní informace; ruku v ruce s redundancí jde i nekonzistence dat, kdy dvě na první pohled stejné kopie, nejsou stejné.
- **Obtížný přístup k datům**. Každý nový požadavek od uživatele vyžadoval speciální vytvořený program. Tím docházelo k časovým prodlevám a nebylo možné rychle a efektivně získávat data z agend.
- **Izolace dat**. Data byla v různých souborech v různých formátech, tudíž bylo obtížné v průběhu času vytvářet nové programy pro manipulaci s těmito daty.
- **Ochrana dat**. Protože aplikační programy byly přímo svázány s definicemi souborů, bylo velmi obtížné ochranu dat zajistit. Přitom ne každý uživatel by měl mít stejná práva k přístupu.
- **Integrita dat**. Tento problém úzce souvisí s konzistencí dat a znamená, že databáze je správná a v souladu s realitou, tj. odráží skutečnost z reálného světa, databáze je aktuální a může být použitelná (Pokorný a Halaška, 2003).

Tyto problémy se s přechodem k databázovému přístupu částečně odstranily. Jak píše Pokorný (2003, s.5) „jádrem databázového přístupu je odtržení definic dat a údržby dat od uživatelských programů. Data už nejsou organizována v jednotlivých souborech, ale

v komplikovanější struktuře, zvané databáze. Databáze se prakticky skládá z datových prvků, vztahů mezi nimi, integritních omezení a schématu. S takto vytvořenou databází lze snadněji manipulovat, modifikovat dle potřeb a jednodušeji vybírat potřebná data.“

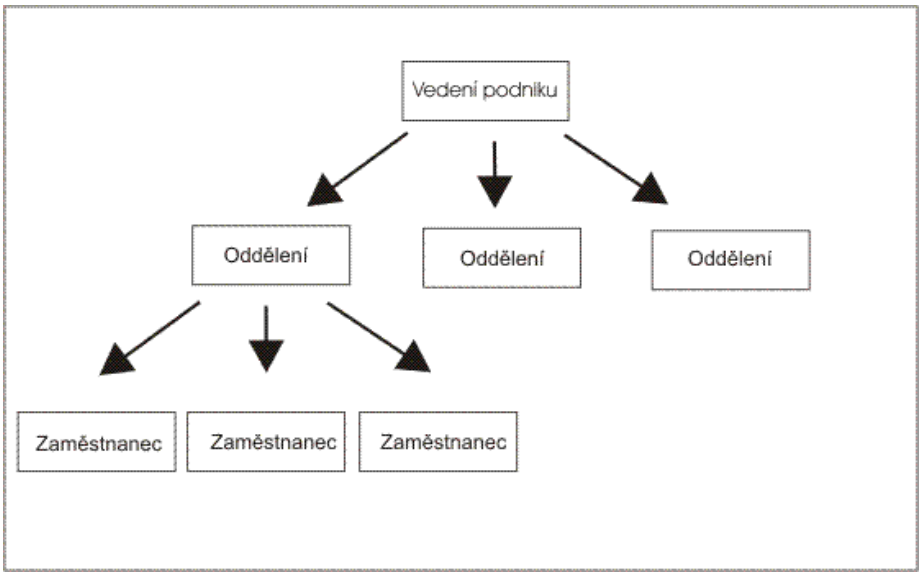
2.2 Databázové modely

Databázové modely, jednoduše řečeno, slouží k modelování struktury databáze. Výsledkem není nic jiného, než databázové schéma, jinými slovy, jistý popis struktury dat. Časem se vyvinuly tři základní modely, které jsou popsány v následujících podkapitolách.

2.2.1 Hierarchický model

První byla hierarchická koncepce, jejíž vznik se datuje k roku 1968 a pojí se k systému IMS vyvinutým firmou IBM (Kaluža a Kalužová, 2012).

Data jsou hierarchicky uspořádaná do podoby obráceného stromu, kdy jedna tabulka slouží jako kořen a vycházející tabulky se nazývají větve (viz obrázek).



Obrázek 2.1 hierarchický model (Ukládání a editace dat)

Vztah je v tomto modelu definován termíny rodič, potomek. Tabulka rodiče může být přidružena k více tabulkám potomků, nicméně tabulka potomka může být přiřazena pouze k jedné tabulce rodiče. Jinými slovy funguje zde vztah 1:N. Nevýhodou této struktury ovšem je, že může obsahovat redundantní data. Model byl hlavně využíván při

ukládání dat na magnetické pásky, to ale zaručovalo pouze sekvenční přístup k datům. Hlavně díky redundanci se tento model přestal v podstatě používat.

2.2.2 Síťový model

Model, jenž je zdokonalením hierarchického modelu, byl definován v roce 1971 na konferenci CODASYL (Conference on Data Systems Languages) a stal se standardem pro prakticky všechny implementace databází DBTG (Database Task Group). Dnes se už síťový model také nepoužívá díky zastaralým principům. Jeho slovník vychází z programovacího jazyka COBOL a definuje pouze binární vztahy typu 1:1 a 1:N mezi dvěma typy záznamů, které se nazývají set. Výhodou tohoto modelu je rychlý přístup k datům, ovšem nevýhodou je nutnost znalosti struktury databáze a nesnadná změna, při které je často nutné změnit i aplikační program pracující s databází.

2.2.3 Relační model

Relační model je ze všech tří zmíněných nejflexibilnější a nejefektivnější. Podle Šimonové (2007) relační databázové systémy vykazují následující charakteristiky:

- Veškerá data se pomyslně dají reprezentovat v pravidelně uspořádaných strukturách s řádky a sloupci, kterým se říká relace.
- Všechny hodnoty v databázi jsou skalární, což znamená, že v každé konkrétní pozici řádku a sloupce dané relace se nachází právě jedna hodnota.
- Operace v databázi se provádějí vždy nad celou relací a jejich výsledkem je opět celá relace. Tomuto mechanismu se někdy říká uzávěr. Díky tomuto mechanismu může být dotaz utvořen nad jiným dotazem, který byl proveden nad určitou relací.

Relační model je charakteristický realizací vazeb pomocí tabulek, v níž každou entitu, která vstupuje do vazby zastupuje primární klíč. Pokud jsou vazby 1:1 , nebo 1:N, není nutné provést spojení pomocí třetí tabulky, ale stačí doplnit na straně N, potažmo na libovolné straně vazby 1:1 další atribut – cizí klíč. Hlavní výhodou modelu je, že později utvořené vazby můžeme realizovat bez zásahu do původní struktury relací.

2.2.4 Objektový a objektově relační model

Objektově orientovaný model rozšiřuje syntaxi jazyků C++, Java aj. a vytváří tak prostor pro databázové programování. Unifikace programovacích aplikací a vývoje

databází je nejdůležitějším přínosem objektového modelu. Aplikace potom vyžaduje menší množství programového kódu a využívají více přirozeného datového modelovaní a programový kód lze lépe pochopit. Díky tomu lze aplikace tvořit efektivněji a jednodušším způsobem.
Podle Šimonové (2002, s. 15) je „přístup k objektově orientovaným databázím kombinací systému objektově orientovaných programovacích jazyků a stabilních systémů. Výhoda těchto databází spočívá v hladkém nakládání s daty, která se nacházejí v databázích a s přechodnými daty v aplikacích."

Objektově relační model, jak tvrdí Šimonová (2002, s. 16) **„**přidává do relačního systému nové objekty jako jádra moderních informačních systémů. Toto nové pojetí začleňuje řízení tradičně postavených dat, složených objektů (systémy časových řad, geoprostorových dat a různá binární media jako např. audio, video…). Na serveru objektově relačního modelu lze vykonávat složité analýzy a operace využívající manipulaci dat. Tyto operace slouží k zobrazení a vyhledávání multimediálních a jiných složitějších objektů."

2.3 Systém řízení báze dat

Jedná se o programové vybavení, díky kterému je zabezpečena centrální správa, ale také vytváření, udržování a modifikování databáze, se kterou když se spojí, tvoří databázový systém. DBMS poskytuje řadu služeb jako transakční zpracování, řízení souběžného přístupu více uživatelů, utajení dat, zahrnutí jazykových prostředků, odolnost vůči chybám a řízení paměti a katalogu dat. Data je také potřeba sdílet, tudíž databázový systém funguje na principu klient/server, kdy veškerá data jsou uložena na serveru (centrální počítač) a přístup k datům je zajištěn přes klienty, pod kterými si lze představit například uživatelské počítače.

S DBMS je také spojena existence dvou typů jazyků:

- **DDL** – jedná se o jazyk pro definici dat a slouží k vytvoření všech definic uživatelských dat potřebných v aplikaci.
- **DML** – jde o jazyk pro manipulaci dat a používá se jak k aktualizaci dat, tak k výběru podle daných kritérií.

Nejznámějším jazykem, který zahrnuje jazyk pro definici i pro manipulaci dat se nazývá SQL, který ale obsahuje jiné podjazyky tvořené například příkazy pro udílení

práv uživatelům. Nejznámější výrobci těchto produktů jsou hlavně Oracle, IBM, nebo Microsoft (Pokorný a Halaška, 2003).

2.4 Datové modelování

Aby byla databáze plně funkční, byla zajištěna integrita a odstraněna redundance dat, je třeba důkladně provést modelování databáze. Obecně se používá hlavně takzvaná tříúrovňová koncepce datového modelování, která zahrnuje sémantickou, konceptuální a logickou úroveň procesu.

2.4.1 Sémantický model

První úroveň odráží modelovanou realitu. Prvky reality se označují jako typy objektů. Jedná se v podstatě o první odraz reality. Metodou, kterou se tyto prvky transformují do typů objektů, je metoda abstrakce. Existují tři typy abstrakce:

- **Klasifikace** – aplikuje se pro identifikaci typů objektů jako základních konstruktorů odrážejících objektivní realitu.
- **Agregace** – definuje nový typ objektu z množiny typů objektů, které se stanou jeho komponentami. Například zaměstnanec se vytvoří sloučením jeho komponent jméno, věk, adresa…
- **Generalizace –** ta definuje vztah podmnožiny mezi výskyty dvou nebo více objektů.

Abstrakce se většinou provádí ve třech krocích. Prvním je identifikace vstupních datových požadavků, kde se vymezí cíle a rozsah řešení datového modelu. Tím se rozumí shromažďování informací z dokumentů, dotazníků, nebo konzultací s budoucími uživateli. Druhým krokem je specifikace typů objektů a jejich charakteristik, kdy se určí typ objektu (například zaměstnanec) a jeho charakteristiky (jméno, příjmení, věk). Posledním krokem je revize struktury, kdy se revidují vytvořené typy objektů. Tím se rozumí odstranění zbytečných charakteristik, rozdělování, nebo slučování typů objektů a odstraňování redundance.

2.4.2 Konceptuální datový model

Principem konceptuálního datového modelu, je jeho vyjádření pomocí grafických nástrojů. První metodou, která byla použita v tomto modelu je metoda E -R diagramu, kterou představil v roce 1976 Peter Chen. Tato metoda je vedle metody diagramu tříd,

která tvoří součást metodiky UML (vytvořena v roce 1995 P. Boochem, J. Rumbaughem a I. Jacobsonem) nejpoužívanější v konceptuálním modelování (Kaluža a Kalužová, 2012).

2.4.2.1 *Entita*

Jak tvrdí Lacko (2003) a Kaluža (2012) jedná se o objekt reálného světa, který je jednoznačně odlišný od ostatních objektů a je schopen nezávisle na ostatních existovat. Takovým objektem může být například *Zaměstnanec, Výrobek,* nebo *Zákazník*. Graficky lze entitu vyjádřit obdélníkem, v jehož horní části se nachází název entity a v dolní části jsou atributy. Nutno podotknout, že si nelze plést entitu a výskyt entity, oba pojmy totiž znamenají něco jiného. Na rozdíl od entity, která byla popsána, výskytem entity se rozumí pouze jeden řádek entity (entitním typem entity *Výrobek* je židle).

Entita může být silná, nebo slabá podle toho jak závisí na existenci jiné entity. Pokud je entita silná, tak svým primárním klíčem nezávisí na žádné entitě, kdežto u slabé entity neexistuje žádný atribut, který by ji identifikoval - primární klíč (Kaluža a Kalužová, 2012), (Šimonová a Panuš, 2007).

2.4.2.2 *Vztah*

Vztah se dá popsat jako vazba mezi dvěma, nebo více entitami. Nejběžnějším vztahem je tzv. **asociativní vztah**. Tento vztah reprezentuje asociace jedné, nebo více entit. Každý tento vztah má tři charakteristiky:

Stupeň

Je to charakteristika, kterou se rozumí počet entit, které se vyskytují v jednom vztahu. Stupeň může být:

- **unární** (rekurzivní) – „Nejvyšší hierarchická úroveň je v takovéto vazbě tvořena jedním prvkem, který je svázaný s prvky o jednu úroveň níže.“ (Lacko, 2003). Proto každý prvek má vazbu na jeden prvek z vyšší úrovně. Toto neplatí pro prvek nejvyšší úrovně. Díky unární relaci lze vyjádřit vztahy typu nadřízený – podřízený (Lacko, 2003).
- **binární** - ve vztahu se vyskytují dvě entity.
- **ternární** - ve vztahu se vyskytují tři entity.

Takzvaný n-nární vztah se v praxi vyskytuje minimálně.

Kardinalita

Kardinalita vyjadřuje počet výskytů entit v jednom výskytu vztahu.

- **1:1** – Každá entita je spojena vztahem nejvýše s jednou další entitou. To podle Lacka (2003, str. 37) znamená, že „každý řádek primární tabulky lze svázat právě s jedním řádkem sekundární tabulky“. V této kategorii vztahů se taky vyskytují takzvané částečné vztahy 1:0 a 0:1, které znamenají, že určitý řádek entity (výskyt entity) není přiřazen k žádnému výskytu entity v druhé tabulce.
- **1**:**N** – Entita E je spojena vztahem s žádnou, nebo více entitami E_x. Tento vztah je velmi důležitý, jelikož eliminuje duplicitní data a nadbytečná data minimalizuje.
- **M**:**N** – Při tomto vztahu nejsou kladeny žádné podmínky a entity mezi sebou mohou být propojeny jakkoli, neexistuje žádné omezení. To obecně znamená, že více řádků entity může být spojeno s více řádky jiné entity. Většina databázových systémů ovšem se vztahy M:N neumí pracovat, proto se tento typ vztahu v reálném světě uskutečňuje pomocí třetí spojovací tabulky. V praxi to znamená, že se vztah rozloží na dva vztahy 1:N (Šimonová a Panuš, 2007), (Lacko, 2003).

Volitelnost

Poslední charakteristikou asociativního vztahu je volitelnost. Ta určuje, zda je vztah povinný ze strany jedné entity, nebo druhé, či nikoliv (Kaluža a Kalužová, 2012).

Po asociativním vztahu je **generický vztah**, pojmenován také generalizace. Podle Kaluži (2012, str. 45) „entita E je generalizací skupin entit E_1, E_2,…,E_n, jestliže každý objekt této skupiny entit je zároveň objektem entity E.“ Entitu E nazýváme supertypem a entity E_1, E_2,…,E_n, jsou subtypy. V obráceném směru se tento vztah nazývá specializace.

2.4.2.3 *Atribut*

Jedná se o funkci, nebo vlastnost přiřazující jednotlivým entitám, nebo vztahům hodnotu. Atributy mohou být jednoduché, nebo složené. Například v entitě

Zaměstnanec jsou atributy *Jméno, Příjmení, Datum narození* (Šimonová a Panuš, 2007), (Lacko, 2003).

2.4.2.4 *Klíč*

Klíč slouží jako identifikátor entity. Jedná se o množinu atributů (nebo také pouze jeden atribut), která jednoznačně určuje výskyt entity, tedy jeden řádek tabulky. Takový identifikátor by měl být vhodně zvolený a měl by splňovat následující kritéria:

- jednoznačnost – jedna hodnota identifikátoru by měla určovat právě jeden výskyt entity,
- úplnost – pro každý výskyt entity by měl existovat právě jeden identifikátor,
- minimální – klíč by neměl být tvořen žádnou podmnožinou, která je jednoznačná, nebo úplná,
- stabilita – identifikátor nesmí během své existence měnit svou hodnotu.

Existují také různé druhy klíčů podle toho, jak se použijí a v jakém jsou vztahu.

- Primární klíč – neboli identifikační klíč se používá v rámci entity a jednoznačně určuje výskyt entity,
- kandidátní klíč – atributy, které lze zvolit jako primární klíče,
- cizí klíč – takhle se označuje atribut, který je v jiné entitě použit jako primární klíč. Tento klíč se primárně využívá k vyjadřování vztahů mezi entitami,
- alternativní klíč - pokud je jeden klíč zvolen jako primární, ale entita má další atributy, které jednoznačně určují entitní typ, ale nejsou zvoleny jako primární, nazývají se kandidátní,
- sekundární klíč – je to nejednoznačná množina atributů, která slouží k vyhledávání a třídění dat, které se nacházejí v entitě (Šimonová a Panuš, 2007).

2.4.2.5 *Doména*

Podle Kaluži (2012, str. 47) se doménou rozumí „množina přípustných hodnot přiřazená jednomu, nebo více atributům. Například množina všech hodnot příjmení (doména) může být přiřazena k atributu příjmení v entitě *Zaměstnanec*, ale též ke stejnojmennému atributu v entitě *Pojištěnec*.“

2.4.3 Logický relační datový model

Nástup relační koncepce usnadnil technologický zlom vyvolaný nástupem osobních počítačů. Relační datové modelování přebírá některé konstrukty z konceptuální úrovně. Mezi ně patří atribut, doména a klíč, nicméně zavádí nový pojem relace.

2.4.3.1 *Relace*

Kaluža a Kalužová (2012, str. 73) definují relaci jako „dvourozměrnou datovou strukturu, která je tvořena záhlavím a tělem relace. Záhlavím relace se rozumí množina dvojic (A_i, D_i), kde atribut A_i je přiřazen právě jedné doméně D_i, pro i = 1 ,2 ,…,n a zároveň A_i musí navzájem být odlišná. Tělo relace je tvořeno množinou n -tic, které jsou množinami dvojic (A_i, v_{ji}), kde A_i je i -tý atribut a dále v_{ji} je j -tá hodnota z domény D_i pro j =1 ,2 ,…,m , kde m je počet n -tic v množině; m je pak kardinalitou a n stupněm relace (pro n =1 se hovoří o unární relaci, pro n =2 o binární, atd.).“

Relace mají tyto základní vlastnosti:

- neexistence duplikátů n -tic
- libovolné pořadí atributů
- libovolné pořadí n -tic
- nerozložitelnost hodnot atributů

První tři vlastnosti znamenají, že tělo relace je v podstatě množinou atributů, poslední vlastnost ukazuje na atomičnost atributů. Tato vlastnost znamená, že atributy nelze dále rozložit, aniž by došlo ke ztrátě informace a výrazně odlišuje relační koncepci od předchozí síťové, ve které atribut může obsahovat skupiny údajů a můžou se zde nacházet opakující se hodnoty. Pokud relace neobsahuje vícehodnotové atributy, lze tvrdit, že je v první normální formě (Kaluža a Kalužová, 2012).

2.4.3.2 *Relační algebra*

Relační model zahrnuje relační algebru a relační kalkul. Podle koncepce, kterou navrhl Codd, databáze získaly aparát predikátové logiky 1. řádu, které se vyrovnají díky své úspornosti ve vyjadřování relačnímu kalkulu. Relační algebra je v podstatě množina operací, které když se aplikují na relace, vracejí opět relaci. Pro tyto operace lze využít množinové operace jako součin, sjednocení, průnik a rozdíl.

Podle Šimonové (2007) se v relační algebře rozlišují operátory:

- **Union** – je to sjednocení a vrací relaci, která obsahuje n-tice, jež se vyskytují v jedné, nebo v obou sjednocovaných relacích.
- **Intersection** – vrací relaci obsahující n-tice, které se vyskytují v obou relacích. Jedná se tedy o průnik.
- **Diference** – zastupuje rozdíl a vrací relaci s n-ticemi, které se vyskytují pouze v první, ale ne v druhé ze specifikovaných relací
- **Product** – česky součin a vrací relaci, která obsahuje všechny možné n-tice vzniklé kombinací dvou n-tic. Jinými slovy lze product nazvat kartézským součinem dvou relací.
- **Restrict** – jedná se o restrikci a vrací n-tice z určené relace, které vyhovují dané podmínce.
- **Project** – vrací relaci, která obsahuje n-tice, které zůstanou po eliminaci určených atributů.
- **Join** – spojování dvou relací vyhovující podmínce (natural, inner, outer).
- **Divide** – dělení.

2.4.3.3 *Normalizace*

Normalizace je proces, kdy dochází k odstraňování anomálií v datovém modelu. Datový model se postupně dekomponuje tak, že se atributy dělí do většího počtu relací, které jsou již normované. Relace se postupně transformují do vyšších normálních forem, až dokud není zajištěna další nedělitelnost.

Normální formy jsou v podstatě soubor pravidel, které by měla relace splňovat, aby práce s jejími daty byla jednodušší. Hlavní normální formy jsou čtyři:

- **První normální forma (1NF)** – „Relace je v první normální formě tehdy, když všechny atributy jsou atomické“ (Lacko, 2003, str. 44). Tato forma vyplývá z definice relace a její základní vlastnosti o nedělitelnosti atributů.
- **Druhá normální forma (2NF)** - Podle Šimonové (2007, str. 63) a Lacka (2003, str. 46) „tabulka splňuje podmínku o druhé normální formě tehdy, splňuje-li první normální formu a každý neklíčový atribut je plně funkčně závislý na

primárním klíči relace.“ Neklíčový atribut musí být závislý na celém primárním klíči.

- **Třetí normální forma (3NF)** – „Relace splňuje 3NF, jestliže je v 2NF a každý neklíčový atribut je netranzitivně závislý na primárním klíči.“ (Šimonová a Panuš, 2007, str. 63)
- **Boyce-Coddova normální forma (BCNF)** – „V této formě je relace, jestliže každý determinant funkční závislosti v relaci je zároveň kandidátním klíčem. Každá relace v BCNF je vždy v 3NF, což naopak neplatí.“ (Šimonová a Panuš, 2007, str. 63). Jestliže nastane, že 3NF není v BCNF, zřejmě za to můžou tyto důvody:
    - v relaci jsou nejméně dva kandidátní klíče,
    - všechny kandidátní klíče jsou složené,
    - existuje překrývající se atribut v kandidátních klíčích.

  Je vidět, že BCNF je striktnější než 3NF (Kaluža a Kalužová, 2012).
- **Čtvrtá normální forma –** Lacko (2003) definuje čtvrtou normální formu jako tabulku, která je ve třetí normální formě a popisuje jeden fakt, nebo souvislost.
- **Pátá normální forma** – „Tabulka je v normální formě tehdy, je-li ve čtvrté normální formě a není do ní možné přidat nový sloupec, případně skupinu sloupců bez toho, aby se rozpadla na několik dílčích tabulek.“ (Lacko, 2003, str. 48)

Čtvrtá a pátá normální forma se vyskytuje zřídka, jelikož představují výjimečné situace, kdy primární klíč má tři a více složek. Obecně převládá názor, že v praxi stačí použít transformaci do BCNF.

Návrh metodiky relačního modelu

E-R diagram, nebo diagram tříd se transformuje do logické relační formy. Vytváří se tzv. předběžné relace, které jsou specifikovány jménem, kandidátními klíči a cizími klíči. Poté se relace zkompletuje doplněním zbývajícími atributy, proběhne normalizace a určení doménových charakteristik. Tato metodika tedy probíhá v následujících krocích:

- **Vytvoření soustavy předběžných relací** – vytvoří se pouze předběžné relace s primárními a cizími klíči. Účelem je, aby model byl přehlednější a byla jasnější vazba mezi relacemi, jelikož neklíčové atributy znepřehledňují model.
- **Přiřazení zbývajících atributů** – v tomto kroku se relacím přiřadí zbývající neklíčové atributy, které byly specifikované na začátku konceptuálního modelování, ovšem doteď se s nimi nepracovalo.
- **Normalizace modelu** – při tomto procesu dochází k odstraňování nedostatků tabulek jako je redundance nebo možnost vzniku aktualizační anomálie. Tím se rozumí nechtěný vedlejší efekt operace nad databází, u kterého může dojít ke ztrátě nebo nekonzistenci dat. V relacích se identifikují všechny funkční závislosti a relace se převedou na BCNF, která je z praktického hlediska nejvhodnější normální forma.
- **Revize konceptuálního modelu** – Díky eventuálnímu rozložení, nebo spojení některých relací v relačním modelování musí dojít také k revizi konceptuálního modelu, kde může docházet k vytváření nových entit a nových vztahů.
- **Specifikace domén** – Tento krok v podstatě přiřazuje všem atributům jejich datový typ, délku, přípustné hodnoty, formát, jedinečnost, přípustnost null hodnot a případně i textový popis. Jinými slovy se vytvoří charakteristiky platných hodnot tvořících domény (Kaluža a Kalužová, 2012).

2.5 Oracle

Firma Oracle vznikla v roce 1977, kdy Larry Ellison, Bob Miner a Ed Oates založili firmu Software Development Laboratories. Firma se přejmenovala na Oracle až poté, co dělala projekt pro CIA se stejným názvem. V současnosti se firma nezaobírá pouze vývojem softwaru pro tvorbu relačních databází, ale mimo to nabízí i celou řadu jiných produktů jako například servery (SPARC, Sun x86, Sun Blade a Sun Netra), operační systémy (Oracle Solaris založený na platformě UNIX, nebo systém Oracle Linux), software na virtualizaci, správu podniku (Oracle Enterprise Manager), úložiště a v neposlední řadě Javu (Oracle, 2013).

Oracle Database je systém báze řízení dat (DBMS), který funguje na principu klient/server. Firma Oracle nabízí několik verzí softwaru, takže je vhodný pro uživatele pracujících ve Windows, či Linux. Poslední verze je Oracle 11g, která oproti předchozí

verzi obsahuje více než 400 nových funkcí. Písmeno „g“ za číslem verze označuje, na kterou oblast se Oracle v dané verzi zaměřil nejvíc (v tomto případě „g“ znamená grid). V Oracle 11g byl hlavně optimalizován úložný prostor pomocí komprese dat a jejich fyzické rozdělení na oddíly. K dalším inovacím patří například fast files, total recall, nebo real application testing, ovšem zabývat se těmito novinkami je na této úrovni zbytečné.

Nejvýznamnějšími firmami, které využívají zpracování dat v gridu je například Amazon, Monster, eHarmony, Bebo, nebo Forecast. Oracle Database je využíván prakticky v každém odvětví od zdravotnictví až po potravinářství (Panský, 2007).

2.5.1 Oracle SQL Developer

Oracle SQL Developer je integrované vývojové prostředí, které zjednodušuje vývoj a správu databází Oracle. Je dostupný ke stažení jako freeware z oficiálních stránek Oracle a mezi jeho hlavní služby patří kompletní end-to-end vývoj PL/SQL aplikací, list pro spouštění skriptů a dotazů, DBA konzoli pro správu databáze, kompletní řešení problému modelovaní dat a možnost importovat databáze v různých formátech přímo do programu. Mezi nesporné výhody patří možnost exportu databáze do formátu PDF ve formě E-R diagramu, nebo jednoduchého popisu jednotlivých tabulek. Mezi další formáty, do kterých je možno exportovat jsou CSV, DDL files, nebo Microsoft XMLA.

2.5.2 PL/SQL

Mezi nejpopulárnější programovací jazyky patří v současnosti zejména Java, PL/SQL, C++, PHP nebo Visual Basic. I přesto, že se jedná o rozdílné jazyky, některé z nich mají některé vlastnosti společné. Například Visual Basic je jazyk procedurální, tedy lineární, což znamená, že začíná na začátku a končí na konci. Jinými slovy každý výraz musí počkat, až skončí předchozí a až po té se dostane na řadu a může vykonat svou činnost. Java, nebo C ++ jsou objektově orientované jazyky. Každý objekt má své atributy. Pokud je třeba změnit nějaký atribut, zavolá se metoda, která tento úkon provede. Tohle je hlavní rozdíl mezi procedurálními jazyky a objektově orientovanými.

PL/SQL se vyznačuje tím, že patří do obou výše zmíněných kategorií. Poprvé se objekty objevily v Oracle 8, nicméně ještě nebyla zajištěna podpora vlastností jako dědičnost, dynamický polymorfismus (dynamická volba metody), nebo typový vývoj. Až ve verzi Oracle 9iR1, kdy se firma Oracle začala snažit tyto nedostatky odstraňovat,

se tyto vlastnosti začaly objevovat. Všechny základní objektově orientované vlastnosti jsou k dispozici od verze Oracle 10g.

Sečteno podtrženo, PL/SQL je vlastně programovací rozšíření jazyka SQL poskytované firmou Oracle výhradně pro Oracle databáze. PL/SQL vyšlo z dalšího programovacího jazyka Ada, ze kterého převzalo některé principy. PL ve zkratce znamená *procedural language* a SQL *structured query language* (Urman, 2007).

2.5.2.1 *Datové typy PL/SQL*

Většinou jsou možnosti a omezení databázových datových typů a typů PL/SQL stejné, nicméně některé mají odlišné možnosti ukládání, proto by se neměly navzájem zaměňovat.

Mezi datové typy PL/SQL patří následující kategorie:

- skaláry
- složené
- odkazy
- LOB (velké objekty)

Skaláry

V skalárním datovém typu je uložena jediná hodnota. Taktéž je lze rozdělit do několika podkategorií, jež zahrnují:

- **Znaky a řetězce** – zahrnují všechno od jediného jednoduchého znaku až po 32K místa zabírající řetězce. Mohou obsahovat písmena, čísla, binární data, čili všechny znaky, které databáze podporuje. Mezi základní znakové a řetězcové typy patří CHAR, NCHAR, NVARCHAR2, RAW, ROWID, UROWID, VARCHAR, VARCHAR2 aj.
- **Číselné typy** – obsahují základní celočíselné datové typy uchovávající celá čísla. Mezi nejběžnější číselné typy patří BINARY_DOUBLE, BINARY_FLOAT (slouží k složitým výpočtům) a NUMBER. Poslední jmenovaný datový typ podporuje celá čísla i hodnoty s desetinnou čárkou.
- **Logické hodnoty** – tato kategorie obsahuje jediný typ, kterým je BOOLEAN. Ten nabývá tří hodnot – true, false, null.

- **Datum a čas** - jsou totožné jako datové typy v databázi. Patří zde DATE, TIMESTAMP a INTERVAL.

Složené typy

Na rozdíl od skalárních typů mají vnitřní komponenty, tudíž mohou obsahovat několik skalárních hodnot nazývané atributy. Jedná se o záznamy, vnořené tabulky, tabulky s indexem a proměnná pole.

Odkazy

Oracle v PL/SQL uchovává v kategorii odkazy dva typy. Těmito typy jsou REF CURSOR a REF. REF CURSOR se nazývá kurzor a lze jím získávat záznamy z procedury, nebo funkce. REF je využíván s objektovými typy. Je to ukazatel na instanci objektu v objektové tabulce, či pohled.

2.5.2.2 *Procedury*

Podle Loneyho (2002, str. 41) se procedura definuje jako „blok příkazů PL/SQL uložená v systémovém katalogu a je volána aplikacemi. Pomocí procedur lze uchovávat často používanou aplikační logiku v rámci databáze."

Procedura nevrací žádné hodnoty, i když tohle není podmínkou a existují určité výjimky. Po uložení dostávají jedinečné jméno a vlastníkem je ten, kdo je píše. Pokud ve vytvářecím skriptu neuvede jiného vlastníka, který proceduru tím pádem vlastní. Procedury se používají například k vynucení zabezpečení dat, ovšem najde se i mnoho dalších oblastí, kde se dají efektivně využít.

2.5.2.3 *Funkce*

Jedná se o podobný blok kódu jako u procedur s tím rozdílem, že funkce volajícímu programu musí vracet hodnotu. Největší rozdíl mezi funkcí a procedurou je povinná klauzule RETURN. Funkce si lze vytvořit vlastní a volat je pomocí SQL, nebo je možno spouštět funkce poskytované přímo platformou Oracle (například funkce SUBSTR, která vrací část řetězce). Při spuštění funkce je ovšem nutno zpracovat její návratovou hodnotu (Loney, 2002), (Urman, 2007).

2.5.2.4 *Balíky*

Balíky jsou logická seskupení procedur a funkcí. Jejich specifikace jsou uloženy v systémovém katalogu, stejně jako jejich obsah. Pokud jsou balíky často využívány

aplikacemi, je nutno zvětšit tabulkový prostor SYSTEM kvůli zvětšení velikosti katalogu.

Využívají se zejména v administrativních úlohách, kdy je třeba řízení procedur a funkcí. Prvky balíků lze definovat jako veřejné nebo soukromé. Uživateli balíku jsou proto veřejné prvky přístupné, zatímco soukromé jsou skryty. Mezi ně patří například procedury volané jinými procedurami v rámci jednoho balíku.

2.5.3 Triggery

Jsou to procedury spouštějící se při specifikované události v databázi. Využívají se například k zvětšení referenční integrity, nebo vynucení dodatečného zabezpečení. Triggery se vyskytují ve dvou typech:

- **Triggery příkazů** – spustí se jednou při každém odpovídajícím příkazu. Používají se v případě, že akce triggeru nezávisí na datech ovlivněných transakcí. Lze například omezit vkládaní do tabulky na stanovené intervaly.
- **Triggery řádků** – Spustí se jednou pro každý řádek tabulky ovlivněný daným příkazem. Jde o opačný příkad jako u triggeru příkazu, tudíž akce triggeru závisí na datech ovlivněných transakcí.

Pro lepší vysvětlení trigger příkazů pro příkaz delete, který má za úkol smazat 10 000 řádků, se spustí pouze jednou, kdežto trigger řádků se spustí desettisíckrát za jednu transakci.

2.5.4 Indexy

V systému Oracle se data vyhledávají pomocí identifikátoru rowID, který je přiřazen ke každému řádku v tabulce. Díky rowID lze zjistit, kde přesně se daný řádek nachází.

„Index je databázová struktura používaná serverem k rychlému vyhledávání řádku tabulky.“ (Loney, 2002, str. 38).

Rozlišují se tři základní typy indexů:

- Indexy clusterů – uchovávají klíčové hodnoty clusterů,
- Indexy tabulek – zde jsou uloženy hodnoty řádků tabulky a fyzické umístění jednotlivých řádků, jinými slovy rowID,

- Bitmapové indexy – jedná se o zvláštní případ indexu, který podporuje dotazy velkých tabulek se sloupci obsahujícími několik rozdílných hodnot.

V Oraclu jsou indexy ukládány pomocí B*tree, který napomáhá k rychlému přístupu ke klíčové hodnotě.

2.6 Java

Jedná se o poměrně nový programovací jazyk, který je v současnosti řazen k nejpoužívanějším a nejrozšířenějším jazykům na světě. Samotný jazyk vychází z jazyka C, ovšem jeho syntaxe je mnohem jednodušší a odbourává některé problémy, které nastávaly v jazycích C a C++. Java postupně dobývá svět hlavně díky tomu, že je nezávislá na jakékoli hardwarové platformě, tím pádem aplikace vyvinuté v Javě lze spustit prakticky kdekoli a v jakémkoli operačním systému. Díky tomu lze vyvíjet jak desktopové, tak serverové aplikace, nebo webové služby. Během své existence se vyvinuly různé „druhy“ Javy, hlavně platforma pro mobilní telefony (ME), desktopy (SE) a distribuované systémy (EE). V současnosti zažívá největší rozmach Java pro mobilní telefony hlavně díky novým technologiím a opensource operačním systémům.

Další výhodou Javy je, že se jedná o objektově orientovaný jazyk. Tento typ jazyku zapouzdřuje data a metody pro jejich změnu do oddělených jednotek, čili objektů. Objekty proto lze klidně použít i v dalších programech (Chapman, 2001).

Java byla vyvinuta firmou Sun Microsystems v roce 1995 a obsahovala knihovnu s pouhými 211 veřejnými třídami. Zlom nastal na konci roku 1998 a začátkem 1999, kdy se přepracovala celá knihovna a došlo k zavedení dalších knihoven, proto počet veřejných tříd stoupl na 1524. Koncepce práce s jazykem se změnila natolik, že se nová verze jazyka a platformy začala označovat jako Java 2 (Pecinovský, 2005).

Nejnovější verzí je v současnosti Java 7, jejíž poslední updaty jsou stále poněkud zranitelné.

2.6.1 Základní prvky Javy

Java se skládá ze tří komponent:

- **Programovací jazyk Java** – Jak bylo popsáno výše, je podobný syntaxí jazyku C++. Hlavními rozdíly mezi Javou a právě C++ je, že v C++ je to programátor, který musí přidělovat paměť, kdežto v Javě je přidělování a uvolňování paměti

automatizováno. V Javě na rozdíl od C++ není umožněna vícenásobná dědičnost, ovšem tenhle zdánlivý nedostatek je kompenzován implementací libovolného počtu rozhraní.

- **Java Virtual Machine (JVM)** – je to základ platformy Java a vyznačuje se tím, že zajišťuje běh kódu na různých platformách. Všechny programy v Javě se totiž musí přeložit do bajtového kódu, což je jazyk interpretovaný JVM. Bajtový kód ovšem nejde spustit přímo, proto každý typ počítače má interpretr (překladač), který převede kód JVM do strojového jazyka konkrétního překladače.
- **Aplikační programovací rozhraní Javy (API)** – jedná se o připravené softwarové komponenty, poskytující standardní způsoby pro čtení a zapisování souborů, práci s řetězci, nebo užívání grafického rozhraní a mnoho jiných. Tyto komponenty se sdružují do balíků (knihoven). To významně ulehčuje práci, protože není třeba psát kód objektů znova a v případě potřeby lze objekty kdykoli použít (Chapman, 2001).

2.6.2 Grafická uživatelská rozhraní

Grafické uživatelské rozhraní je obrazové rozhraní k programu. Obvykle bývá vybaveno tlačítky, nabídkami a dalšími komponentami z nabídky knihovny Swing. Jazyk Java obsahuje dvě grafická rozhraní. Staré rozhraní známé pod zkratkou AWT, neboli Abstract Windowing Toolkit a nové pod názvem Swing GUI, které je rychlejší a flexibilnější, než AWT.

K vytvoření grafického uživatelského rozhraní jsou potřeba čtyři základní prvky:

- **Komponenty** – nejedná se o nic jiného, než o tlačítka, popisky, textová pole a další položky. Všechny komponenty mají společnou sadu metod určenou k nastavení velikostí, barev, stylu písma a podobně. V balíku *javax.swing* má každá komponenta metody určené ke své funkci.
- **Kontejner** – všechny komponenty jsou uspořádány do kontejneru. Jedná se o JPanel a mnohem složitější JFrame s ohraničením a záhlavím. Opět lze tyto kontejnery najít v balíku *javax.swing*.
- **Správce rozvržení** – po přidání komponenty do kontejneru správce rozvržení určí, kam se komponenta umístí. V Javě je poskytováno šest správců rozvržení

(anglicky layout manager) a každý z nich rozmisťuje komponenty jiným způsobem. Programátor ovšem může tyto layout managery libovolně měnit. Stejně jako kontejnery i komponenty, jsou správci rozvržení obsaženi v balíku *javax.swing*. Mezi hlavní patří *BorderLayout, BoxLayout, FlowLayout* a *GridLayout*

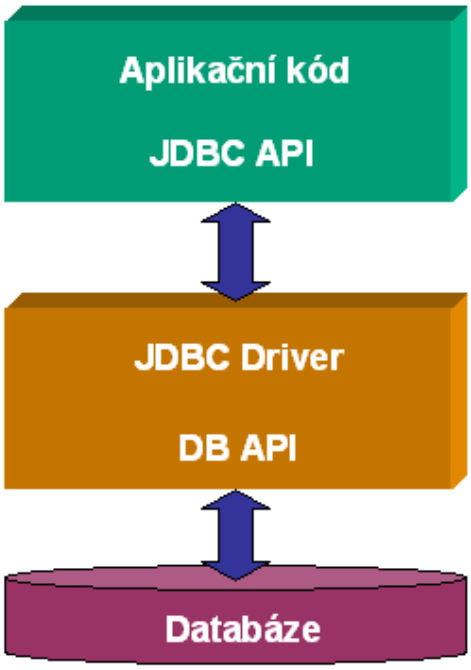
- **Obsluha událostí –** po klepnutí tlačítka myši, nebo po vstupu z klávesnice, musí být provedena požadovaná akce. Těmito akcemi se vytvoří událost, která je objektem Javy. Události jsou obsluhovány vytvořením listenerů sledující určitý typ událostí a provedou jistou metodu, pokud událost nastane. Listenery implementují rozhraní naslouchání specifikující jména metod obsluh událostí určených k ošetření určité události. Toto rozhraní se nachází v balíku *java.awt.event* (Chapman, 2001).

2.6.3 JDBC

Aby bylo možno pracovat s relačními databázemi, poskytuje Java rozhraní JDBC, což znamená Java Database Connectivity. „Jako ovladač JDBC je označována implementace rozhraní Driver a dalších rozhraní z balíčku java.sql a dalších pomocných tříd, které zajišťují přístup k databázi.“ (VŠE, 2005)

JDBC neslouží pouze k přístupu k relačním databázím, ale i k jiným formátům dat, které se ukládají do sloupcové podoby. Jedná se například o CSF soubory a mnoho dalších.

JDBC bylo inspirováno ODBC standardem, který navrhla firma Microsoft. ODBC je čistě C-čkové aplikační rozhraní bez jakéhokoli objektového základu, navíc nepřehledné a nestrukturované. Proto Microsoft od ODBC upustil a v .NET je přístup k datům řešen přehledněji a nabízí podobnou škálu možností, jako JDBC. Na obrázku 2.2 je možno vidět architekturu JDBC.



2.2 Architektura JDBC (Šeda, 2003)

JDBC specifikace rozpoznává čtyři typy ovladačů, kterými jsou:

- **Most JDBC-ODBC** – ovladač využívá lokální ODBC ovladač a přistupuje k němu právě přes JDBC-ODBC most. ODBC ovladače se liší od výrobce databáze, proto je jejich aplikace složitá a je nutno instalovat lokální DLL knihovny, které je nutno synchronizovat na aktuální databázi. Proto se tento typ hodí hlavně pro testování v prostředí Windows nebo pro aplikace využívající JSP a Servlety (Šeda, 2003).
- **Připojení prostřednictvím kódu nativního klienta pro přístup k síti** – ovladač tohoto typu je napsán v Javě pomocí nativního kódu. Komunikuje s klientským softwarem používaného databázového systému, proto musí být na klientské stanici nainstalován příslušný software.
- **Připojení prostřednictvím vrstvy aplikačního serveru** – tento typ ovladače je napsán kompletně v Javě. Posílá požadavky na server serverové komponentě ovladače odpovídající za převod do formátu dané databáze. Díky tomuto řešení se může databáze měnit bez nutnosti změny klienta, pouze na serveru se zprovozní jiná serverová komponenta pro připojení.

- **Přímé spojení s databází** – stejně jako předchozí typ, taktéž tento typ ovladače je napsán celý v Javě. Komunikuje přímo s databází, snadno se ovládá a může být součástí balíčku aplikace.

„U každého ovladače je potřeba v dokumentaci zjistit, jakou verzi specifikace JDBC podporuje. Možnosti použití jsou též ovlivněny vlastními možnostmi databáze, ke které se připojuje.“ (VŠE, 2005)

2.6.4 NetBeans IDE

Jedná se o komplexní freewarové vývojové prostředí, které poskytuje prvotřídní a komplexní podporu pro nejnovější technologie Java. Netbeans poskytuje příjemné uživatelské prostředí, kdy zdůrazňuje zdrojový kód jak syntakticky, tak sémanticky. Nabízí šablony kódu, tipy pro kódování a také refactoring nástroje. Editor nepodporuje pouze programovací jazyk Java, nýbrž také C/C++, XML, HTML, PHP, JSP, Javascript a mnohé další. Editor lze nainstalovat na různých platformách, které podporují Javu od Windows, přes Linux až po Mac OS. Je rozšiřitelný, a proto do něj lze doinstalovat, nebo vytvořit pluginy, které lze později využívat pro vývoj aplikací. Jelikož se tato práce zabývá desktopovou aplikací, jsou využity pouze základní možnosti NetBeans zahrnující také připojení k databázi (NetBeans IDE, 2011).

3 Analýza současného stavu ve firmě

Firma Medové pečivo sídlící v Ostravě vznikla už v roce 1994 a zabývá se výrobou dekorativních perníků k různým událostem jako například svátky, narozeniny, svatby, nebo Velikonoce a Vánoce. Nabízí také vytvoření dekorativních perníků pro firemní nebo reklamní účely. Firma působí hlavně v České republice, nicméně některé objednávky směřují i do zahraničí, zejména na Slovensko nebo také do Německa.

V současnosti jsou všechna data uložena v excelovských tabulkách a ve Wordu. Jedná se o tabulky s výrobky, zákazníky a objednávkami. Toto řešení sice poskytuje všechny potřebné informace, ovšem případné změny v datech musí být prováděny složitě a uživatel, neznaje práci s Excelem a jeho funkcemi, nemusí tyto operace provést správně, a proto může docházet k chybám v tabulkách. Podobných situací lze najít více, proto by bylo vhodné tyto procesy do jisté míry zjednodušit a zautomatizovat.

Pokud jsou do tabulek vkládány nové záznamy, třeba i díky nepozornosti může dojít k redundanci dat. Záznamy v Excelu mohou na někoho působit nepřehledně, výběr dat pomocí filtrů může být složitý a celkově samotný Excel může pro někoho být nepřívětivý pro běžné užívání. Všechny tyto nedostatky vyřeší komplexní databáze výrobků, zákazníků, objednávek a vystavených faktur, kterou díky uživatelskému rozhraní lze snadno obsluhovat. Samozřejmě lze zakoupit software, který by tato kritéria také splňoval, ovšem takovéto programy jsou většinou nákladné a to je hlavní problém, který je třeba brát v potaz. I proto je vhodné vytvořit vlastní aplikaci, která bude zdarma a v podstatě bude splňovat základní potřeby pro firmu. Takovými potřebami je rychlá a přehledná evidence faktur a objednávek a jejich tisk. K aplikaci se bude přistupovat pouze z jednoho počítače, na kterém je také vyvíjena, proto není třeba řešit otázku přístupu více uživatelů k datům v databázi v jeden moment.

4 Realizace a implementace databázové aplikace

4.1 Tříúrovňová koncepce databáze

Aby bylo datové modelování co nejpřesnější, je vhodné použít víceúrovňové zpracování. Jestliže je datový model vytvářen rovnou a objekty jsou implementovány přímo do příslušného softwaru, programátor nemůže tušit, které prvky reality jsou důležité a budou vyjádřeny relacemi, nebo které atributy mají být použity a přiřazeny. Samotný atribut se někdy může stát relací, nebo naopak. Tyhle problémy jsou většinou odstraněny pomocí víceúrovňového zpracování datového modelu. Podle Kaluži (2012, str. 16) je nejvhodnější tříúrovňový model, zahrnující sémantickou úroveň, která má tato specifika:

- „identifikace datových požadavků vyplývajících z obsahu řešeného projektu,
- vytvoření výchozí datové struktury jako bezprostředního odrazu modelované objektivní reality,
- priorita co nejúplnějšího (a zároveň relevantního) popisu reality před propracovaností vnitřní struktury modelu.“

Informace se z reality mohou získávat různými způsoby. Mezi nejběžnější patří zkoumání formulářů, dotazníků a jiných textových materiálů, nebo také rozhovorem s respondenty za účelem zjistit, které objekty jsou pro budoucí vývoj důležité. V tomto případě jako zdroj informací posloužily hlavně excelovské tabulky a informace od respondenta, ze kterých se vyextrahovaly následující objekty.

4.1.1 Specifikace sémantického modelu

V této úrovni jsou vyjmenovány všechny objekty a jejich charakteristiky, které později budou převedeny do konceptuálního modelu. Jedná se o abstrakci reality, kdy objekty vyjadřují modelovanou realitu. V tomto modelu se ještě nebere v potaz integrita, primární klíče a vztahy mezi objekty, modelují se pouze ty prvky, které jsou pro pozdější vývoj podstatné.

Název objektu: Výrobky
Popis: Údaje o výrobku a ceně
Charakteristiky: název, velkoobchodní cena, maloobchodní cena

Název objektu: Zákazník
Popis: Informace o zákazníkovi, potažmo firmě a její sídle
Charakteristiky: název firmy, město, ulice, stát

Název objektu: Položky objednávky
Popis: Objekt nesoucí informace o množství objednaných výrobků
Charakteristiky: objednané množství, skutečné množství, velkoobchodní cena, maloobchodní cena, poznámka

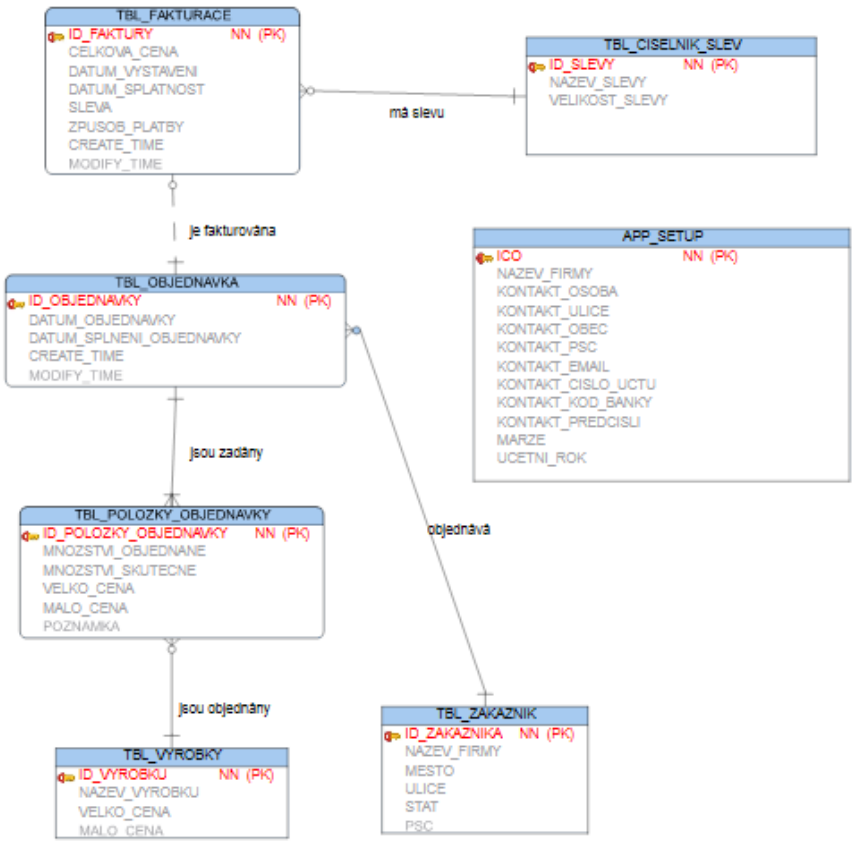
Název objektu: Objednávka
Popis: Údaje o objednávce zadané zákazníkem
Charakteristiky: datum objednávky, datum splnění objednávky

Název objektu: Fakturace
Popis: Specifikace vystavení faktury
Charakteristiky: celková cena, datum vystavení, datum splatnosti, sleva, způsob platby

Název objektu: Číselník slev
Popis: Velikost slevy na objednávku
Charakteristiky: velikost slevy, název slevy

4.1.2 Konceptuální model

Principem tohoto kroku je převést všechny typy objektu do entit a znázornit jejich vztahy mezi sebou. V praxi to znamená určit kardinalitu mezi jednotlivými tabulkami (zda je vztah 1:N , 1:1 , nebo M:N), volitelnost vztahů, případně generalizaci nebo specializaci. Nejjednodušším způsobem je tyto vztahy znázornit graficky v E-R diagramu. Základními komponenty jsou entita a vztah a jeho vlastnosti (viz str. 12). Na obrázku 4.1 je konceptuální model znázorněný pomocí E-R diagramu. Tento model neřeší logickou strukturu databáze, to znamená, že v modelu se neřeší jak cizí klíče a spojení tabulek, ale hlavně žádná konkrétní databázová koncepce. Logická struktura se řeší až následně v logickém relačním datovém modelu.



4.1 Grafické znázornění konceptuálního modelu (Crow's foot notace)

4.1.3 Vytvoření relačního modelu

V tomto kroku se provádí transformace konceptuálního modelu do relačního. Právě zde je třeba datový model dořešit tak, aby jej bylo možno transformovat do definičního jazyka databáze. Entita je transformována do jedné relace přepisem i s odpovídajícími atributy. V relaci se určí primární klíč, posléze cizí klíč, který určí vztahy mezi

jednotlivými relacemi. U vztahu 1:N se primární klíč z jedné tabulky přidá do tabulky druhé, kde se označí jako cizí klíč. Vztah 1:1 je třeba určit, zda je povinný, nebo nepovinný. Pokud je povinný na jedné straně, zapíše se obdobně jako 1:N , pokud je oboustranně povinný, lze obě entity v tomto vztahu modelovat do jedné relace.

4.1.3.1 *Tabulka tbl_vyrobky*

Tato tabulka je, dá se říci, nejdůležitější, jelikož obsahuje data o výrobcích, které lze zákazníky objednat. Obsahuje data o velkoobchodní ceně, což znamená cena bez marže obchodníka a maloobchodní cena, ke které je marže již připočtena. Dále je zde název výrobku, jehož maximální délka je 40 znaků, a primární klíč, který je určen atributem ID_VYROBKU, jenž je datového typu number a logicky nesmí obsahovat prázdné hodnoty. V následující tabulce je celkový přehled datových typů jednotlivých atributů a specifikací domén:

TBL_VYROBKY (ID_VYROBKU #, VELKO_CENA, MALO_CENA, NAZEV)

Název	***Datový typ***	***Délka***	***Klíč***	***Unique***	***Null***	***Formát***
ID_VYROBKU	Number		PK	Ano	Ne	
VELKO_CENA	Number			Ne	Ne	
MALO_CENA	Number			Ne	Ne	
NAZEV	Varchar2	40		Ne	Ne	

Tabulka 4-1 Specifikace domén tbl_vyrobky

4.1.3.2 *Tabulka tbl_zakaznik*

V tabulce se nacházejí informace o zákazníkovi, který si někdy objednal zboží u firmy. Její atributy jako název, město, ulice a stát jsou znakového datového typu varchar2, pouze atribut PSC je typu char a jeho omezení je na pět znaků. Atribut ID_ZAKAZNIKA je datového typu number a slouží jako primární klíč.

TBL_ZAKAZNIK (ID_ZAKAZNIKA #, NAZEV_FIRMY, MESTO, ULICE, PSC, STAT)

Název	***Datový typ***	***Délka***	***Klíč***	***Unique***	***Null***	***Formát***
ID_ZAKAZNIKA	Number	10	PK	Ano	Ne	9999999999
NAZEV_FIRMY	Varchar2	20		Ne	Ne	A (20)
MĚSTO	Varchar2	40		Ne	Ne	A (40)
ULICE	Varchar2	20		Ne	Ne	A (20)
PSC	Char	5		Ne	Ne	9 (5)
STAT	Varchar2	20		Ne	Ne	A (20)

Tabulka 4-2 Specifikace domén tbl_zakaznik

4.1.3.3 *Tabulka tbl_polozky_objednavky*

Tato tabulka už je poněkud složitější. Mimo primární klíč ID_POLOZKY_OBJEDNAVKY obsahuje i cizí klíče ID_VYROBKU a ID_OBJEDNAVKY, které znázorňují vztah 1:N s tabulkami *tbl_vyrobky* a *tbl_objednavka* (viz obrázek 4.1). Dále se zde nachází údaje o objednaném množství výrobku a skutečně vyrobeném množství, protože ne vždy se nutně tyto dva údaje musí shodovat (zákazník si objedná třeba 6 kusů jistého výrobku, nicméně z nějakých důvodů se vyrobí a zaúčtuje pouze 5). Dále je zde velkoobchodní a maloobchodní cena. Tyto dva atributy se zdají být na první pohled redundantní k stejně nazvaným atributům v tabulce *tbl_vyrobky*, účel těchto dvou atributů ale spočívá v tom, že pokud by došlo ke změně ceny výrobku v tabulce *tbl_vyrobky*, mělo by to vliv na všechny již vystavené faktury v historii. Proto tyto dva atributy slouží k tomu, aby nedocházelo k tomuto nežádoucímu jevu. Dále je zde atribut, sloužící k vložení nějaké poznámky k položkám objednávky.

TBL_POLOZKY_OBJEDNAVKY (ID_POLOZKY_OBJEDNAVKY #, ID_OBJEDNAVKY (FK), ID_VYROBKU (FK), MNOZSTVI_OBJEDNANE, MNOZSTVI SKUTECNE, VELKO_CENA, MALO_CENA, POZNAMKA)

Název	***Datový typ***	***Délka***	***Klíč***	***Unique***	***Null***	***Formát***
ID_POLOZKY_OBJEDNAVKY	Number	10	PK	Ano	Ne	9999999999
ID_OBJEDNAVKY	Number	10	FK	Ne	Ne	9999999999
ID_VYROBKU	Number	10	FK	Ne	Ne	9999999999
MNOZSTVI_OBJEDNANE	Number	10		Ne	Ne	9999999999
MNOZSTVI_SKUTECNE	Number	10		Ne	Ano	9999999999
VELKO_CENA	Number	10		Ne	Ne	9999999999
MALO_CENA	Number	10		Ne	Ne	9999999999
POZNAMKA	Varchar2	100		Ne	Ano	A (100)

Tabulka 4-3 Specifikace domén tbl_polozky_objednavky

4.1.3.4 *Tabulka tbl_objednavka*

Tato tabulka slouží jako hlavička konkrétní objednávky, ve které jsou shromážděny základní data o objednávce a jejich zadavatelích. Jako primární klíč slouží atribut ID_OBJEDNAVKY, který musí být samozřejmě jedinečný a není přípustná null hodnota. Cizí klíč reprezentuje ID_ZAKAZNIKA, který odkazuje na tabulku *tbl_zakaznik* a určuje vztah 1:N. Dalšími atributy jsou DATUM_SPLNENI_OBJEDNAVKY a DATUM_OBJEDNAVKY které jsou datového typu date a jak název napovídá, určují data, kdy byla objednávka zadána a do kdy se má splnit. Posledními atributy jsou CREATE_TIME a MODIFY_TIME, které obsluhují triggery a slouží k zjištění, kdy byla faktura zadána do databáze a kdy modifikována. Tyto dva atributy se nevyužijí přímo v aplikaci, ale slouží pouze pro kontrolu dat.

TBL_OBJEDNAVKA (ID_OBJEDNAVKY #, ID_ZAKAZNIKA (FK), DATUM_SPLNENI_OBJEDNAVKY, DATUM_OBJEDNAVKY)

Název	***Datový typ***	***Délka***	***Klíč***	***Unique***	***Null***	***Formát***
ID_OBJEDNAVKY	Number	10	PK	Ano	Ne	9999999999
ID_ZAKAZNIKA	Number	10	FK	Ne	Ne	9999999999
DATUM_SPLNENI_OBJEDNAVKY	Date			Ne	Ano	DD.MM.YYYY
DATUM_OBJEDNAVKY	Date			Ne	Ano	DD.MM.YYYY
CREATE_TIME	Date			Ne	Ne	DD.MM.YYYY
MODIFY_TIME	Date			Ne	Ano	DD.MM.YYYY

Tabulka 4-4 Specifikace domén tbl_objednavka

4.1.3.5 *Tabulka tbl_fakturace*

Tabulka slouží k ukládání dat o vystavených fakturách za objednávku. Jako ve všech předchozích tabulkách je řádek tabulky jednoznačně určen primárním klíčem ID_FAKTURY, který zároveň udává i číslo faktury. Spojení s tabulkou *tbl_objednavka* je zajištěno pomocí ID_OBJEDNAVKY s kardinalitou 1:1 . Nabízí se možnost, že by se tyto dvě tabulky mohly spojit v jednu a tento na první pohled zbytečný vztah by zmizel. Nicméně objednávka nemusí být z nějakého důvodu realizována, nebo za ni nejsou inkasovány peníze, například u sponzorského daru, a proto faktura není vystavena. ID_OBJEDNAVKY je tedy cizí klíč, ale je to unikátní atribut, tudíž je zabezpečeno, že se nesmí vyskytovat v této tabulce více než jednou a je tak vytvořen právě vztah s kardinalitou 1:1 . Atributy DATUM_VYSTAVENI a DATUM_SPLATNOSTI určují data, kdy byla faktura vystavena a kdy má splatnost, atribut ZPUSOB_PLATBY ukazuje, jak bude faktura zaplacena a nabývá hodnot převodem, nebo hotově. Toto omezení bude lepší vytvořit až v samotné aplikaci. CELKOVA_CENA je číselného datového typu a je to celková suma, která musí být zaplacena. Tento atribut bude počítán také až samotnou aplikací a poté ukládán do databáze. Pomocí ID_SLEVY, který slouží jako cizí klíč je vytvořeno spojení s tabulkou *tbl_ciselnik_slev* a z této tabulky se načte do atributu SLEVA velikost slevy. Je to obdobný případ jako v tabulce *tbl_vyrobky* a *tbl_polozky_objednavky*, kdy

při změně ceny by se změnila i celková cena. Taktéž zde by se celková cena při změně slevy změnila, proto je třeba tuto podmínku ošetřit. Stejně jako v *tbl_objednavka* posledními dvěma atributy jsou CREATE_TIME a MODIFY_TIME.

TBL_FAKTURACE (ID_FAKTURY #, ID_OBJEDNAVKY (FK), ID_SLEVY (FK), CELKOVA_CENA, DATUM_VYSTAVENI, DATUM_SPLATNOSTI, SLEVA, ZPUSOB PLATBY, CREATE_TIME, MODIFY_TIME)

Název	***Datový typ***	***Délka***	***Klíč***	***Unique***	***Null***	***Formát***
ID_FAKTURY	Char	6	PK	Ano	Ne	A (6)
ID_OBJEDNAVKY	Number	10	FK	Ano	Ne	999999999
ID_SLEVY	Number	10	FK	Ne	Ne	999999999
CELKOVA_CENA	Number	10		Ne	Ano	999999999
DATUM_VYSTAVENI	Date			Ne	Ano	DD.MM.YYYY
DATUM_SPLATNOSTI	Date			Ne	Ano	DD.MM.YYYY
SLEVA	Number	10		Ne	Ano	999999999
ZPUSOB_PLATBY	Varchar2	20		Ne	Ano	A (20)
CREATE_TIME	Date			Ne	Ano	DD.MM.YYYY
MODIFY_TIME	Date			Ne	Ne	DD.MM.YYYY

Tabulka 4-5 Specifikace domén tbl_fakturace

4.1.3.6 *Tabulka tbl_ciselnik_slev*

Tato jednoduchá tabulka slouží jako číselník všech slev, které lze uplatnit na jednotlivou fakturu. Je tvořena primárním klíčem ID_SLEVY a atributem, který určuje výšku slevy, tedy VELIKOST_SLEVY. Výše slevy je dána procentem (vyjádřeno pomocí desetinného čísla), aby bylo možno efektivně výslednou procentuální slevu odečíst od celkové ceny. Tento výpočet bude taktéž probíhat v samotné aplikaci.

TBL_CISELNIK_SLEV (ID_SLEVY #, VELIKOST_SLEVY)

Název	***Datový typ***	***Délka***	***Klíč***	***Unique***	***Null***	***Formát***
ID_SLEVY	Number	10	PK	Ano	Ne	9999999999
VELIKOST_SLEVY	Number	10		Ne	Ne	9999999999
NAZEV_SLEVY	Varchar2	40		Ne	Ne	A (40)

Tabulka 4-6 Specifikace domén tbl_ciselnik_slev

4.1.3.7 *Tabulka app_setup*

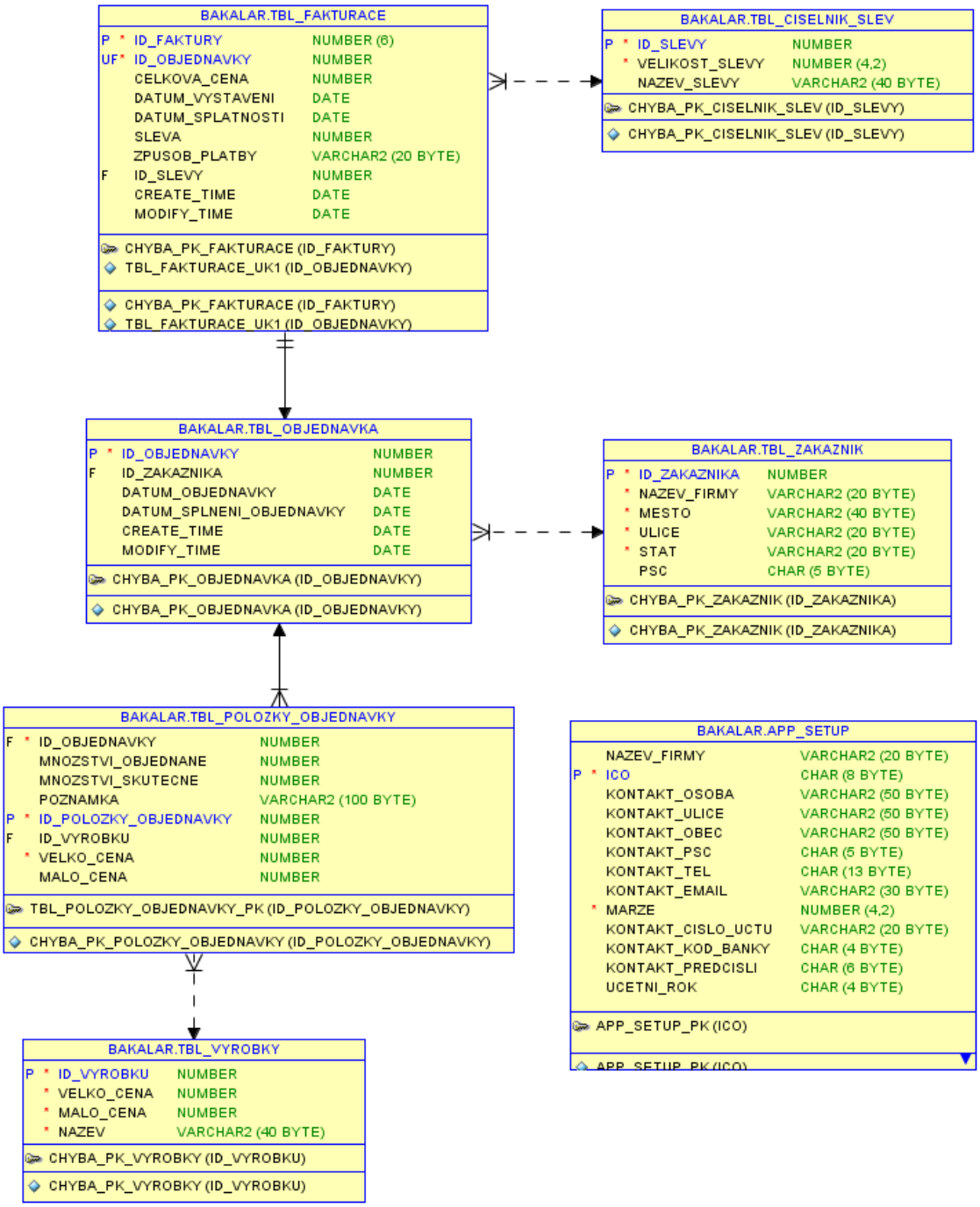
Tabulka *app_setup* je specifická v tom, že není propojena s žádnou předchozí tabulkou, ale slouží pouze pro uchování atributů potřebných pro nastavení aplikace jako jsou informace o firmě, pro kterou je databáze konstruována. Čili ukládá název firmy, její adresu, IČ, kontaktní telefon, email, číslo účtu a jméno vlastníka. Navíc je zde zvláštní atribut MARZE, která slouží k plošnému přepočítávání velkoobchodních cen na ceny maloobchodní. Není tedy nutno jednotlivě přepočítávat každou položku, ale stačí změnit marži a trigger změní ceny všech výrobků v tabulce *tbl_vyrobky*.

APP_SETUP (ICO #, NAZEV_FIRMY, KONTAKT_OSOBA, KONTAKT_OBEC, KONTAKT_ULICE, KONTAKT_PSC, KONTAKT_TEL, KONTAKT_EMAIL, KONTAKT_PREDCISLI, KONTAKT_CISLO_UCTU, KONTAKT_KOD_BANKY, MARZE, UCETNI_ROK)

Název	***Datový typ***	***Délka***	***Klíč***	***Unique***	***Null***	***Formát***
ICO	Char	8	PK	Ano	Ne	A (8)
NAZEV_FIRMY	Varchar2	20		Ano	Ne	A (20)
KONTAKT_OSOBA	Vardar2	50		Ano	Ne	A (50)
KONTAKT_OBEC	Varchar2	50		Ano	Ne	A (50)
KONTAKT_ULICE	Varchar2	50		Ano	Ne	A (50)
KONTAKT_PSC	Char	5		Ano	Ne	A (5)
KONTAKT_TEL	Char	13		Ano	Ne	A (13)
KONTAKT_EMAIL	Varchar2	30		Ano	Ne	A (30)
KONTAKT_PREDCISLI	Char	6			Ano	9 (6)
KONTAKT_CISLO_UCTU	Char	10			Ano	9 (10)
KONTAKT_KOD_BANKY	Char	4			Ano	9 (4)
MARZE	Number			Ano	Ne	999999999
UCETNI_ROK	Char	4			Ne	A (4)

Tabulka 4-7 Specifikace domén app_setup

Na obrázku 4.2 je graficky znázorněn logický model i s cizími klíči, tudíž zde lze pozorovat, jakým způsobem jsou tabulky propojeny. V každé tabulce je vedle názvu atributu i jejich datový typ a vlevo od něj je značení, zda se jedná o primární klíč (P), cizí klíč (F), nebo unique hodnota (U). Tento E-R diagram je generován přímo SQL Developerem.



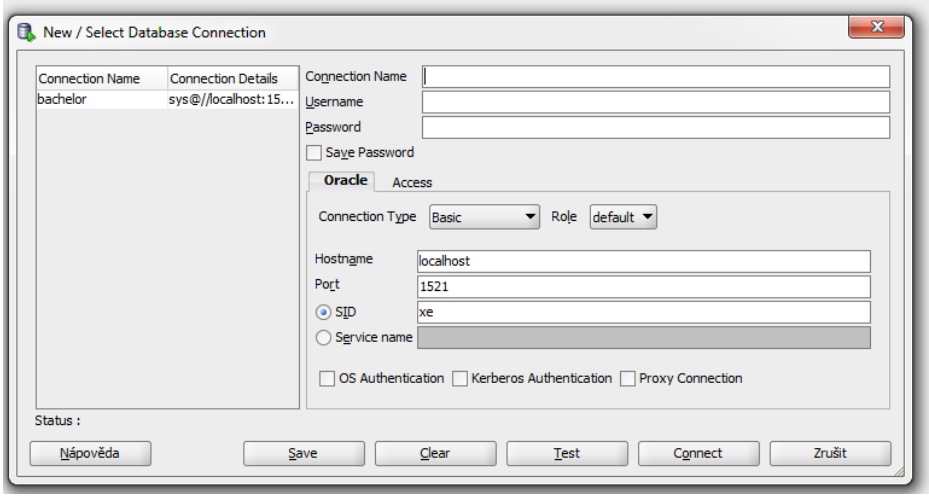
4.2 Grafické znázornění logického modelu (Crow's Foot notace)

4.2 Transformace modelu do DDL v prostředí SQL Developeru

4.2.1 Připojení k databázi

Aby bylo možno vytvořit svoji vlastní databázi, je nutné provést nové připojení v SQL Developeru. Do řádku Connection Name se zadává jméno databáze, do pole Username uživatelské jméno a pole Password znamená heslo. Při vytváření nového připojení je potřeba se přihlásit jako „sys“, čili administrátor, aby byla dostupná všechna práva na

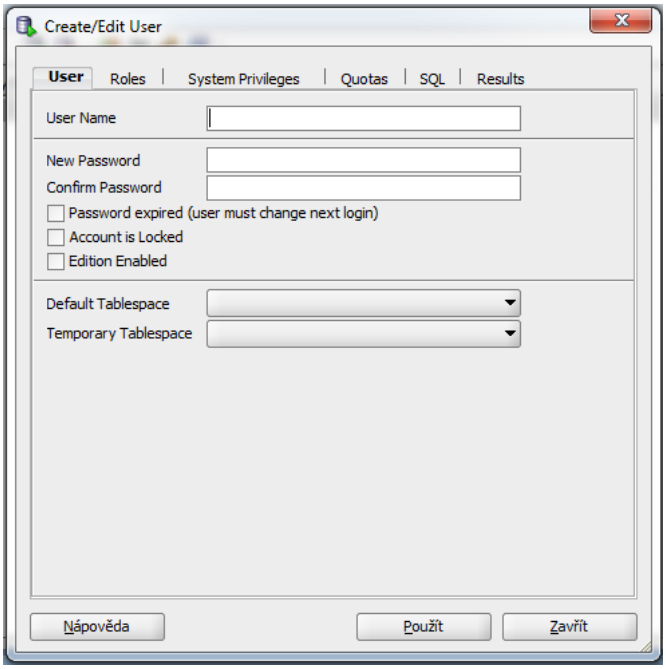
správu databáze a později bylo možno vytvořit nového uživatele. Pro účely této databáze není nutné vyplňovat nic dalšího a už nic nebrání k vytvoření nového spojení.



4.3 Vytvoření nového připojení

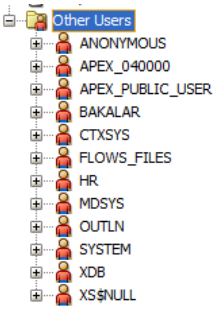
4.2.2 Vytvoření nového uživatele

Protože je nevhodné vytvářet databázi pod systémovým přihlášením, je lepší založit nového uživatele a tomu posléze přidělit práva. Obrázek 4.4 znázorňuje okno, ve kterém se právě nový uživatel zakládá. V této práci byl vytvořen uživatel s názvem BAKALAR s místem pro vytváření tabulek v Users. Tento krok se provede ve výběru pole Default Tablespace. Pro lepší a pohodlnější manipulaci byla tomuto uživateli přidělena většina z administrátorských práv. V liště System Privileges lze nastavit, zda bude mít uživatel možnost vytvářet, měnit a mazat tabulky, triggery, procedury, sekvence a další objekty databáze.



4.4 Vytvoření nového uživatele

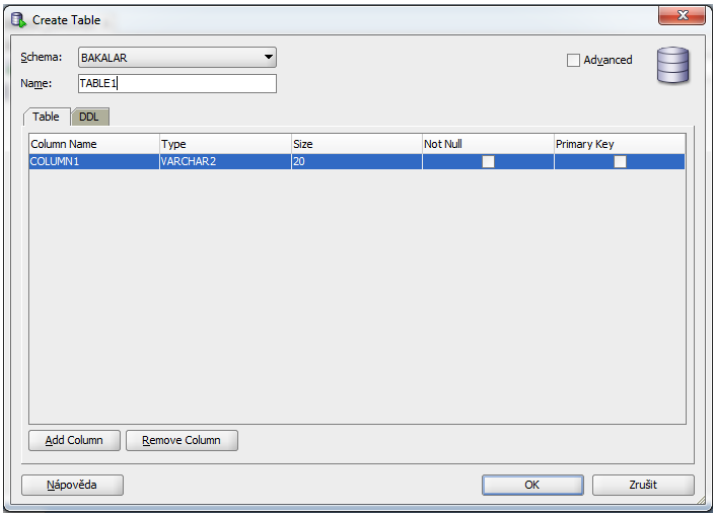
Jak lze pozorovat na obrázku 4.5 ve složce Other Users se objeví nově vytvořený uživatel s přiděleným místem v databázi, kde si lze vytvořit vlastní databázi.



4.5 Další uživatelé databáze

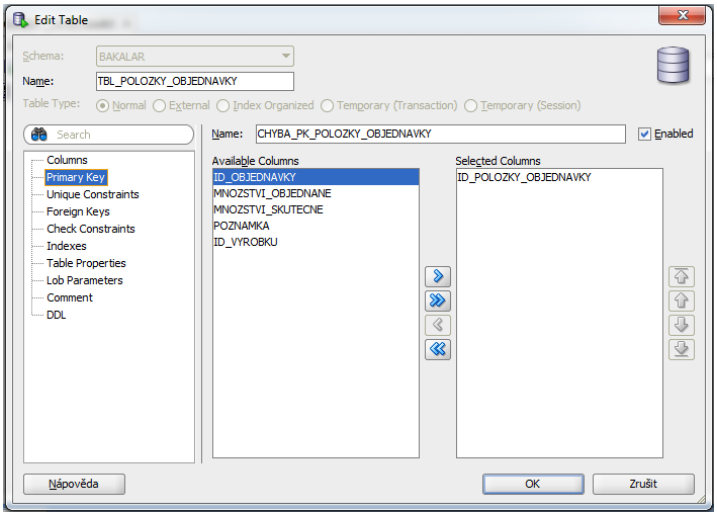
4.2.3 Vytvoření tabulek

Nejčastěji se tabulky vytvářejí pomocí jazyka DDL a to pomocí příkazů CREATE, ALTER a DROP TABLE. Nicméně software SQL Developer nabízí tvorbu tabulek i bez použití SQL příkazů, tudíž je možné použít vytváření a editaci nabízenou přímo programem. Toto řešení je jednodušší a intuitivnější, než samotné SQL příkazy, ovšem v některých případech je výhodnější použít přímo jazyk SQL. Editor tabulek také generuje samotný SQL kód sloužící pro náhled. V poli Schema lze nastavit uživatele, ke kterému se tabulka vytváří, a který může přidávat, upravovat nebo mazat sloupce. Jak je vidět na obrázku 4.6., vytváření tabulek přes okna umožňuje zobrazit celkový přehled sloupců v tabulkách.

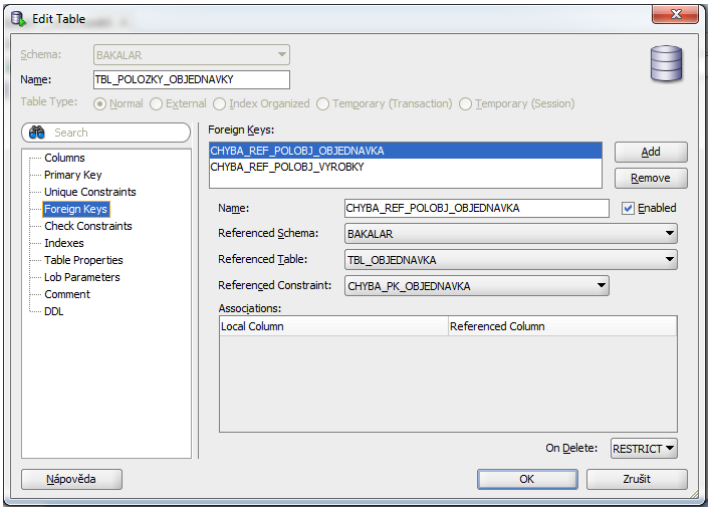


4.6 Vytváření tabulky

V tomto okně lze také vytvořit primární klíče a reference, ovšem tento postup autor příliš nedoporučuje. Pokud ještě není vytvořena tabulka, která má být ve vztahu, může dojít k chybovým hlášením, proto je lepší problém klíčů vyřešit pozdější editací tabulek, která je znázorněna obrázky 4.6 a 4.7. Tvorba cizího klíče je poněkud složitější, ale ve skutečnosti se jedná pouze o název primárního klíče, u kterého se postupně z polí vybere odpovídající „Referenced Schema“ (BAKALAR), tabulka, na kterou má cizí klíč odkazovat a v poli Associations se intuitivně vyplní Local Column a Referenced Column.



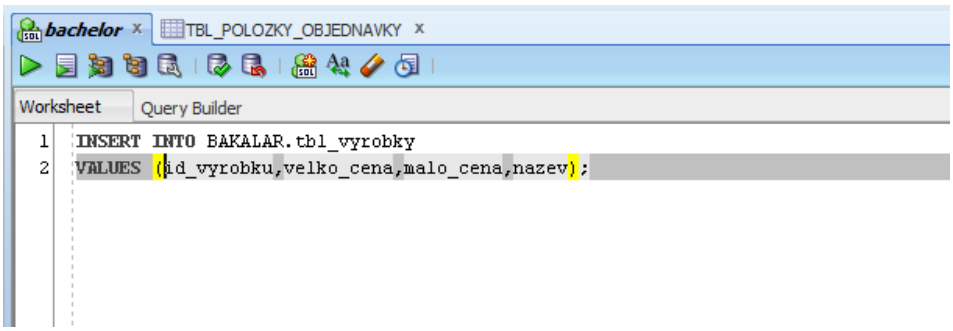
4.7 Editace tabulky a tvorba primárního klíče



4.8 Editace tabulky a tvorba cizího klíče

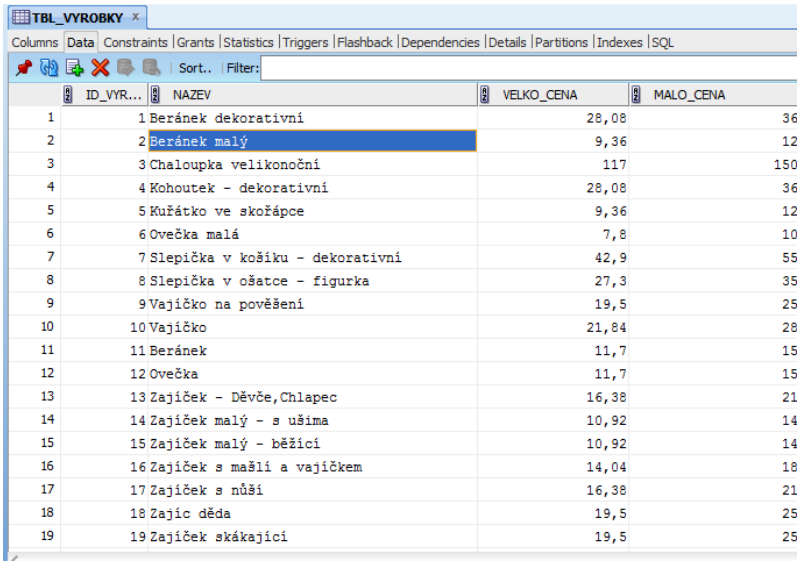
4.2.4 Vkládání dat do tabulek

Stejně jako u vytváření tabulek, kdy je možno použít SQL příkazy, tak i u vkládání dat si lze vybrat, zda data budou vkládána přes editor, nebo pomocí příkazu Insert. Pokud si někdo zvolí zadávání dat pomocí Insertu, spustí se SQL Worksheet, kde lze zapsat příslušný kód (viz obrázek 4.9), který lze následně spustit. Tyto SQL příkazy je možno uložit na disk , kdy se daný kód exportuje do formátu sql. Tento způsob je ale poněkud neefektivní a zdlouhavý, jelikož se musí zadávat data po řádcích, nehledě na syntaktické chyby, které může programátor nechtěně vytvořit.



4.9 Insert pomocí SQL příkazu

Proto je mnohem jednodušší a uživatelsky příjemnější vkládat data pomocí editoru, který znázorňuje obrázek 4.10. Po dokončení editace dat v tabulce je třeba operaci potvrdit a pokud je vše v pořádku, v konzoli se vypíše hlášení „Commit successful“ a Insert je dokončen. Aby se programátor nemusel starat o vyplňování pole ID_VYROBKU, které slouží jako primární klíč a je to jenom určitá number hodnota bez širšího významu, lze tento krok zautomatizovat pomocí vytvoření triggeru a sekvence, která potom automaticky doplňuje hodnoty primárního klíče.



TBL_VYROBKY

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. Filter:

	ID_VYR...	NAZEV	VELKO_CENA	MALO_CENA
1	1	Beránek dekorativní	28,08	36
2	2	Beránek malý	9,36	12
3	3	Chaloupka velikonoční	117	150
4	4	Kohoutek - dekorativní	28,08	36
5	5	Kuřátko ve skořápce	9,36	12
6	6	Ovečka malá	7,8	10
7	7	Slepička v košíku - dekorativní	42,9	55
8	8	Slepička v ošatce - figurka	27,3	35
9	9	Vajíčko na pověšení	19,5	25
10	10	Vajíčko	21,84	28
11	11	Beránek	11,7	15
12	12	Ovečka	11,7	15
13	13	Zajíček - Děvče,Chlapec	16,38	21
14	14	Zajíček malý - s ušima	10,92	14
15	15	Zajíček malý - běžící	10,92	14
16	16	Zajíček s mašlí a vajíčkem	14,04	18
17	17	Zajíček s nůší	16,38	21
18	18	Zajic děda	19,5	25
19	19	Zajíček skákající	19,5	25

4.10 Insert pomocí editoru tabulky

Vkládání dat přímo v databázi pomocí DML jazyka se provádí pouze u tabulek *tbl_vyrobky*, *tbl_zakaznik* a *tbl_ciselnik_slev*, Do dalších tabulek se budou data vkládat až pomocí aplikace vyvíjené v programovacím jazyku Java.

4.2.5 Triggery a sekvence

Databázový systém Oracle na rozdíl od SQL Serveru nebo například databáze FoxPro, čili databázových systémů od firmy Microsoft, neumí automaticky inkrementovat hodnoty primárních klíčů. Proto se musí tento problém ošetřit pomocí sekvencí a triggerů, které danou sekvenci spouští. Tímto způsobem se vyřeší automatické zvyšování hodnoty primárního klíče o určitou hodnotu. Tento postup je samozřejmě vhodný pouze pro ty primární klíče, které nevyjadřují žádnou hodnotu, nebo nemají žádnou funkci, ale slouží opravdu pouze k identifikaci. Sekvence a následné triggery jsou vytvořeny pro každou tabulku v databázi pomocí SQL příkazů v SQL worksheet i přesto, že lze využít editoru stejně jako u tabulek, nicméně v tomto případě je jednodušší rovnou příkaz napsat. Navíc editor triggeru vytvoří pouze začátek a konec triggeru, samotné tělo mezi klauzulemi begin a end už musí být napsáno programátorem.

4.2.5.1 *Trigger a sekvence tabulky tbl_fakturace a tbl_objednavka*

Tyto dva triggery, vytvořené nad tabulkami *tbl_fakturace* a *tbl_objednavka* zajišťují také vytváření atributu CREATE_TIME a MODIFY_TIME, kdy se při vytvoření řádku, nebo změně v řádku zaznamená datum, kdy byl tento úkon proveden. V ostatních tabulkách triggery obstarávají pouze spouštění sekvencí. Nad tabulkou *tbl_fakturace*

sekvence není vytvořena, protože generování primárního klíče bude ošetřeno až samotnou aplikací a tento krok je podrobně popsán v podkapitole 4.3.2.5.

Sekvence je obecně pojmenována seq_nazev_pk, podle primárního klíče tabulky, ve které je vytvářena. Minimální hodnota klíče je 1 , maximální 99999, u každého nového řádku se zvyšuje o 1 a začíná na čísle jedna. Hodnota CACHE se používá zejména při logování, ve statistikách, nebo v tabulkách, kde je PK generován sekvencí, což je právě tento případ. Při vytváření si lze vybrat, zda bude sekvence bez cache, čili s hodnotou NOCACHE, nebo zda cache přiřadíme. Pokud vybereme CACHE n , Oracle automaticky načte poslední použité číslo a rezervuje pro sebe n hodnot, které může daný proces volně používat. Z toho plyne i výkonová výhoda – vícenásobné použití sekvence bez nutnosti přistupovat často na medium. Hodnoty NOORDER a NOCYCLE udávají, že hodnoty primárního klíče mohou být generovány bez řazení a nesmí se opakovat.

Tyto dvě sekvence, stejně jako triggery, jsou v podstatě úplně stejné, liší se pouze tím, že trigger nad tabulkou s fakturacemi nespouští žádnou sekvenci. Jiný je také název triggeru, názvy tabulek a primárních klíčů. Pro příklad poslouží sekvence a trigger nad tabulkou *tbl_objednavka*. První klauzule insert znamená, že pokud jsou vkládána data, spustí se dotaz na hodnotu sekvence, který se poté vloží do atributu primárního klíče a do hodnoty atributu CREATE_TIME se vloží aktuální datum. Druhá podmínka update pro modifikaci dat pouze zajišťuje, že do nové hodnoty atributu MODIFY_TIME se vloží aktuální datum a administrátor tak má přehled, kdy byla data modifikována, respektive kdy byly objednávky, potažmo fakturace zadány do databáze.

```
CREATE SEQUENCE  "BAKALAR"."SEQ_ID_OBJEDNAVKY"  MINVALUE 1 MAXVALUE 99999
INCREMENT BY 1 START WITH 541 CACHE 20 NOORDER  NOCYCLE ;
create or replace
TRIGGER bakalar.trg_objednavka BEFORE
  INSERT OR
  UPDATE ON bakalar.tbl_objednavka FOR EACH ROW BEGIN NULL;
  IF inserting THEN
    SELECT seq_id_objednavky.NEXTVAL INTO :new.id_objednavky FROM DUAL;
    :new.create_time:=sysdate;
  END IF;
  IF updating THEN
    :new.modify_time:=sysdate;
  END IF;
END;
```

4.2.5.2 *Trigger tabulky APP_SETUP*

Jediný trigger, který nezajišťuje spouštění sekvencí je trigger s názvem UPDATE_MARZE, který je vytvořen nad tabulkou *app_setup* a zajišťuje připočítávání marže u ceny výrobků v tabulce *tbl_výrobky* na základě výše marže. Pokud se v tabulce *app_setup* změní atribut MARZE, ke všem cenám výrobků, které jsou uloženy pod atributem VELKO_CENA se tato marže přičte a uloží se do atributu MALO_CENA.

```
CREATE OR REPLACE
TRIGGER BAKALAR.update_marze AFTER
  UPDATE OF MARZE ON BAKALAR.APP_SETUP DECLARE p _marze NUMBER;
  BEGIN
    SELECT marze INTO p _marze FROM BAKALAR.app_setup;
    UPDATE BAKALAR.tbl_vyrobky SET malo_cena=velko_cena*p _marze;
  END;
```

4.2.5.3 *Ostatní triggery a sekvence*

Ostatní triggery pouze spouštějí sekvence nad tabulkami a vyjma výše zmíněných tabulek jsou stejné, liší se pouze názvem a primárními klíči. Pro příklad poslouží sekvence a trigger tabulky *tbl_vyrobky*.

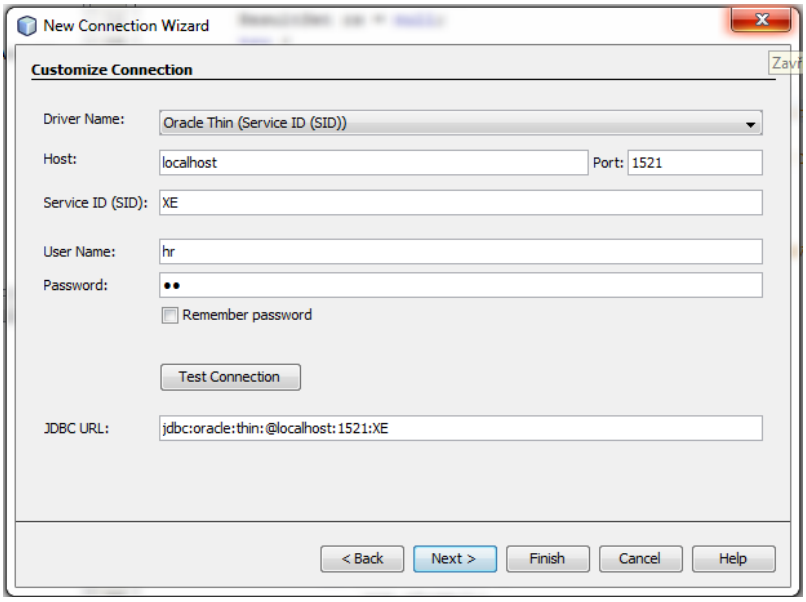
```
CREATE SEQUENCE "BAKALAR"."SEQ_ID_VYROBKU" MINVALUE 1 MAXVALUE 999999
INCREMENT BY 1 START WITH 64 CACHE 20 NOORDER NOCYCLE ;
CREATE OR REPLACE
TRIGGER BAKALAR.trg_vyrobky
BEFORE INSERT ON bakalar.tbl_vyrobky
FOR EACH ROW
BEGIN
IF inserting THEN
    SELECT seq_id_vyrobku.NEXTVAL INTO :new.id_vyrobku FROM DUAL;
    END IF;
END;
```

4.3 Návrh a popis komponent aplikace

Databázová aplikace je vyvíjena jako desktopová, což znamená, že běží na straně desktopového počítače nebo na laptopu. Pracuje primárně s lokálními daty a používá lokálně instalované nástroje, jedná se tudíž o opak webové aplikace. V této části práce jsou popsány základní komponenty aplikace, jejich funkce a připojení aplikace k databázi pomocí vývojového prostředí NetBeans.

4.3.1 Připojení k databázi pomocí JDBC

Aby bylo možno pracovat v aplikaci s databází, je třeba ji připojit pomocí k tomu určenému driveru, který zajistí spojení. V popisovaném programu je použit driver Oracle – Thin, který je dostupný na oficiálních stránkách Oracle. Balíček ovladačů obsahuje také driver Oracle – OCI, je pak na programátorovi, který použije k vytvoření spojení.



4.11 Nové připojení k databázi

Ovladač je poté třeba importovat do seznamu knihoven. Pokud knihovna není explicitně v nabídce, je nutno ji vytvořit. V tomto případě se musí vytvořit knihovna, která je pojmenována MyLibrary, ve které se nachází příslušný jar soubor (ojdbc6.jar). Toto je poslední krok při vytváření spojení s databází a nyní je možno z vývojového prostředí NetBeans manipulovat s daty.

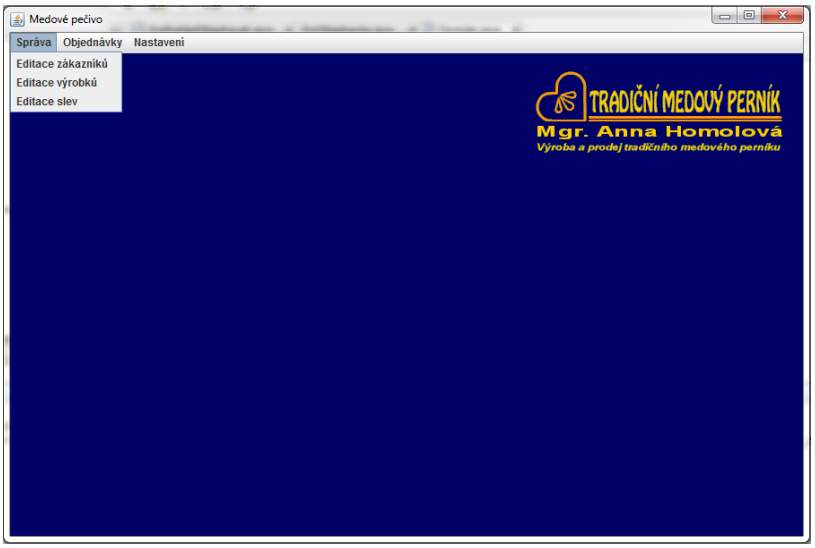
4.3.2 Realizace databázové aplikace v programovacím jazyce Java

Aplikace je navržena jako okenní a je vytvořena pomocí knihovny Swing. Hlavním principem této aplikace jsou okna, pomocí nichž se obsluhují jednotlivé tabulky v databázi.

4.3.2.1 *Úvodní formulář*

Po spuštění aplikace se objeví startovací okno s menu, které má za úkol spouštět další okna. Zároveň s otevřením tohoto okna se automaticky vytvoří spojení s databází. První záložkou je „Správa“, díky které lze spustit tabulky s přehledem výrobků, firem, nebo

číselník slev. Druhá záložka s názvem „Objednávky“ obsluhuje samotnou tvorbu objednávky, přehled zadaných objednávek a následné fakturování. V poslední záložce „Nastavení“ lze spustit okno s informacemi o firmě, které později slouží k vytvoření hlavičky faktury. Horní lišta je vytvořena pomocí komponent JMenuBar, potažmo JMenuItem, což je položka jednotlivé karty. Další okna jsou poté spouštěna pomocí těchto nabídek.



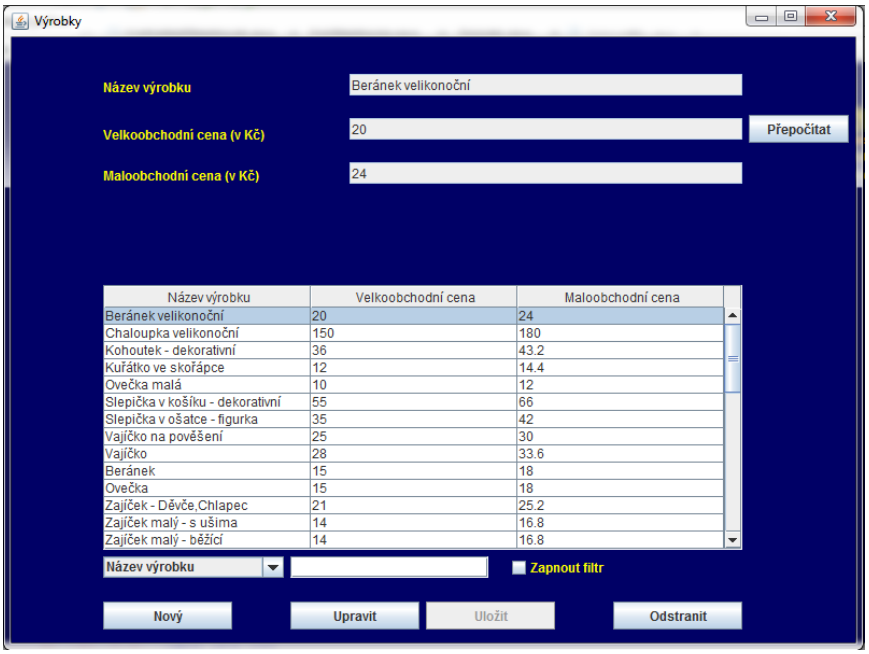
4.12 Hlavní okno aplikace

Úvodní formulář obsahuje také výše zmíněný podformulář s názvem frmNastaveni, který obsahuje pouze komponenty JTextField a tlačítko Uložit změny. Po stisku tohoto tlačítka se veškeré provedené změny uloží do databáze. Při změně marže se přepočítají pomocí triggeru všechny ceny výrobků v tabulce *tbl_vyrobky*. V tlačítku funguje podmínka, která ověřuje, zda byly všechny hodnoty zadány ve správném formátu. Pokud jsou všechna kritéria splněna, ukáže se uživateli MessageDialog s hlášením o úspěšném uložení hodnot a naopak. Vkládání hodnot je zajištěno pomocí SQL příkazu, který je nejprve poskládán do textového řetězce String a poté je takto vytvořený řetězec příkazu proveden obecnou metodou rawSql(), jejíž kód je obsažen v příloze č. 1, která je uložena ve třídě DbManager (viz níže) obsluhující spojení a manipulaci s daty v databázi pomocí aplikace. Tento způsob provádění SQL příkazů je použit v celé aplikaci.

4.13 Podformulář Nastavení

4.3.2.2 *Formuláře číselníků*

Tato skupina formulářů je v podstatě identická, jelikož obsluhují tabulky, které slouží k manipulaci s daty v nich uloženými. Prvním formulářem je FrmZakaznik obsluhující číselník firem, které patří k zákazníkům firmy, druhý formulář s názvem FrmVyrobky slouží jako přehled výrobků a jejich cen, které firma vyrábí a má je v nabídce a díky poslednímu formuláři s názvem FrmCiselnikSlev lze manipulovat se slevami, které firma poskytuje zákazníkům při splnění určitých kritérií. Protože princip těchto tří formulářů je téměř identický, jejich fungování je objasněno na formuláři s výrobky (obrázek č. 4.14).



4.14 Formulář FrmVyrobky

Hlavní komponentou tohoto formuláře je tabulka JTable, která zobrazuje hodnoty z databáze. Tato tabulka je obsluhována pomocí obecné třídy MyTableModel, jež je podtřídou AbstractTableModel. Ta zajišťuje aktualizování komponenty JTable při každém SQL příkazu pomocí metody fireTableDataChanged(). Takto je obsluhována většina tabulek v celé aplikaci až na některé výjimky, které jsou specifikovány v dalších kapitolách práce. Výhodou Table Modelu je možnost si jej modifikovat k vlastním potřebám a jedná se v podstatě o nejpoužívanější a asi nejvhodnější metodu k práci s tabulkami a databázemi.

Pomocí čtyř dolních tlačítek se provádí insert, update a delete z tabulky. Celý SQL příkaz se skládá v metodě provedAkci(), která pomocí podmínky switch-case, jež rozhoduje podle proměnné o tom, který příkaz má být proveden. Obdobně jako ve formuláři s nastavením se i zde skládá příkaz do stringového řetězce a hodnoty z JTextFieldů se pomocí metody getText() vkládají do řetězce. Poté je příkaz vykonán znovu pomocí metody rawSql() a tabulka se aktualizuje.

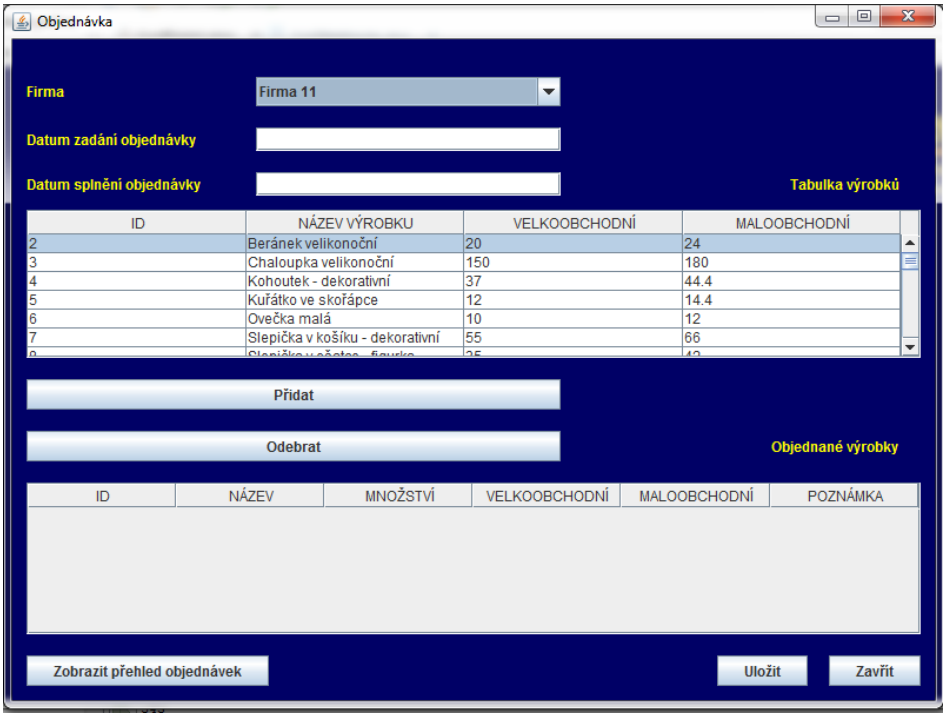
Pod hlavní tabulkou se nachází také filtr, který slouží k tomu, aby bylo možno zobrazit pouze záznamy podle určitých kritérií. Z komponenty JComboBox se vybere sloupec, ve kterém se má spustit filtr v textovém poli se zadá podmínka a samotný filtr se spustí pomocí CheckBoxu. Samotný filtr je prováděn pomocí SQL příkazu Select, kdy je do klauzule Where uložen textový řetězec z textového pole pomocí metody getText(). Tato podmínka je implementována do TableModelu díky parametru cWhere, který pozná, zda je nutno do výběru z tabulky, který provádí přidat i klauzuli Where.

Jediným rozdílem mezi číselníkem výrobků a ostatními číselníky je tlačítko „Přepočítat“, jež má za úkol přepočítat velkoobchodní cenu na maloobchodní. Jinými slovy přidat marži z tabulky *app_setup* k ceně. Jako ve většině případů při práci s textovými poli tlačítko načte pomocí metody getText() hodnotu z textového pole s velkoobchodní cenou a vynásobí ji hodnotou marže. Ta je vybrána pomocí metody getMarze() (viz příloha č. 2) z třídy DbManager. Po vynásobení takto vytažené marže a ceny z textového pole se nová hodnota převede na string a uloží se do druhého pole. Poté je záznam připraven na uložení do databáze.

4.3.2.3 *Formulář FrmObjednavka*

K jedné z nejdůležitějších částí celého programu patří formulář objednávek. Ten slouží k zadávání data objednávky, data splnění, zákazníka, který objednávku zadal a množství objednaných výrobků. Základním prvkem jsou dvě tabulky, díky kterým se

ukládají do databáze objednané výrobky a jejich množství, textová pole s datem, roletka s výpisem firem a dvě tlačítka obsluhující ukládání záznamu do databáze (viz obrázek č. 4.15). Při spuštění formuláře z hlavního menu se automaticky vytvoří řádek v databázi, konkrétně v tabulce *tbl_objednavka* a jeho id se uloží do proměnné, aby s ním bylo možno i nadále pracovat. Stejné id_objednávky funguje jako cizí klíč v tabulce *tbl_polozky_objednavky*, do které jsou ukládány produkty, které byly objednány.



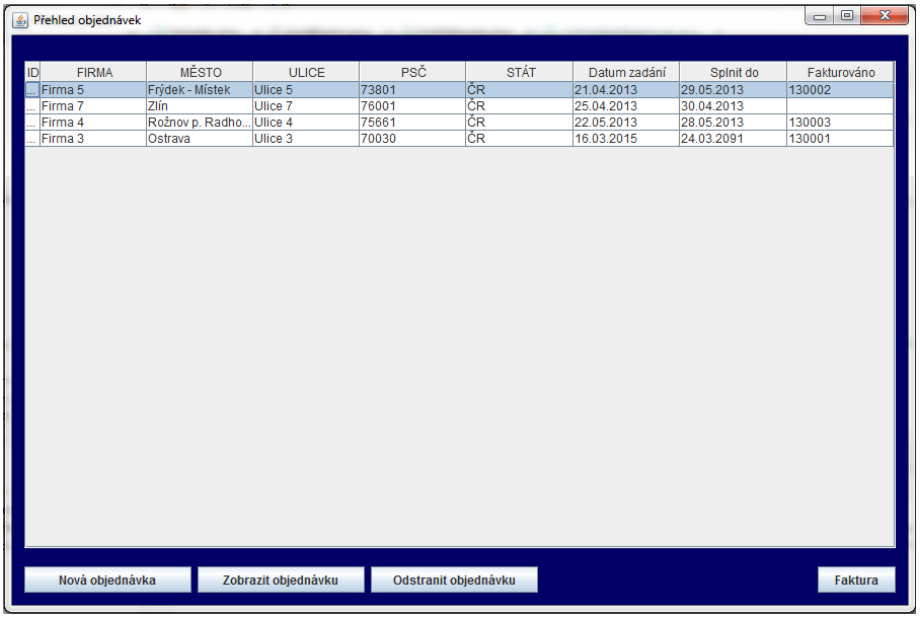
4.15 Formulář objednávky

Naplnění tabulky výrobků funguje obdobně jako u číselníků pomocí TableModelu. Rozdíl nastává u tabulky položek objednávky. Jelikož tabulka *tbl_polozky_objednavky* neobsahuje název výrobku, ale pouze jeho id a proto, že tabulka by bez názvu byla jaksi neúplná, musí být JTable naplněna pomocí příkazu Select hodnotami z dvou tabulek. AbstractTableModel neumožňuje plnění tabulek resultSetem, proto je použita knihovna rs2xml.jar, která má metody umožňující naplnění tabulek právě resultSetem. V tomto konkrétním případě se vytvoří instance třídy TableModel které se přiřadí resultSet pomocí metody resultSetToTableModel(rs), kdy rs je proměnná ResultSet, do které je načten resultSet vytvořený v metodě rawSqlQuery() (příloha č. 1) v DbManageru. Jedná se o podobnou metodu jako rawSql() s tím rozdílem, že rawSql() nepodporuje provádění příkazů Select.

Celá objednávka se uloží pomocí tlačítka „Uložit“. Toto tlačítko v sobě obsahuje řetězec s příkazy Update objednávky a Update položek objednávky a objednávky samotné, které se stiskem tlačítka vykonají. Ke komponentě JCombobox s firmami je vytvořena samostatná třída s proměnnými id a název firmy, díky které je možno zobrazit název firmy, ovšem do Update tabulky umožňuje odeslat id. Objednávku lze stornovat pomocí tlačítka „Storno“, které smaže vytvořený záznam objednávky při inicializaci formuláře a vytvořené záznamy výrobků s id objednávky z tabulky *tbl_polozky_objednavky*. Tlačítko „Zobrazit přehled objednávek“ zavře aktuální okno a otevře nové s přehledem objednávek, které je popsané v následující kapitole.

4.3.2.4 *Formulář FrmPrehledObjednavek*

Z tohoto formuláře se lze dostat buď na vytvoření nové objednávky, zobrazení už objednávky vytvořené nebo na vystavení faktury pomocí tlačítek v dolní části formuláře (obr č. 4.16).



4.16 Formulář s přehledem objednávek

Tlačítky „Nová objednávka“ a „Zobrazit objednávku“ se spouští formulář FrmObjednavka s parametrem pIdObj, což je id objednávky. Pokud je parametr roven nule, tak probíhá inicializace formuláře totožně jako při spouštění pomocí hlavního formuláře a komponenty JMenuItem popsaného v předchozí kapitole. Zobrazení už existující objednávky funguje na stejném principu, nicméně tlačítko pošle parametr do formuláře s objednávkou, který má hodnotu id vybrané objednávky v tabulce. Komponenty ve formuláři jsou poté naplněny daty, které mají hodnotu právě poslaného id. Tlačítko „Odstranit objednávku“ odstraní záznamy z tabulky s položkami

objednávky, kde je totožné id objednávky a záznam objednávky z tabulky *tbl_objednavka.* Po odstranění se spustí metoda fillData() (viz příloha č. 3), která obnoví tabulku ve formuláři pomocí příkazu Select a metody resultSetToTableModel(rs).

4.3.2.5 *Formulář FrmFakturace*

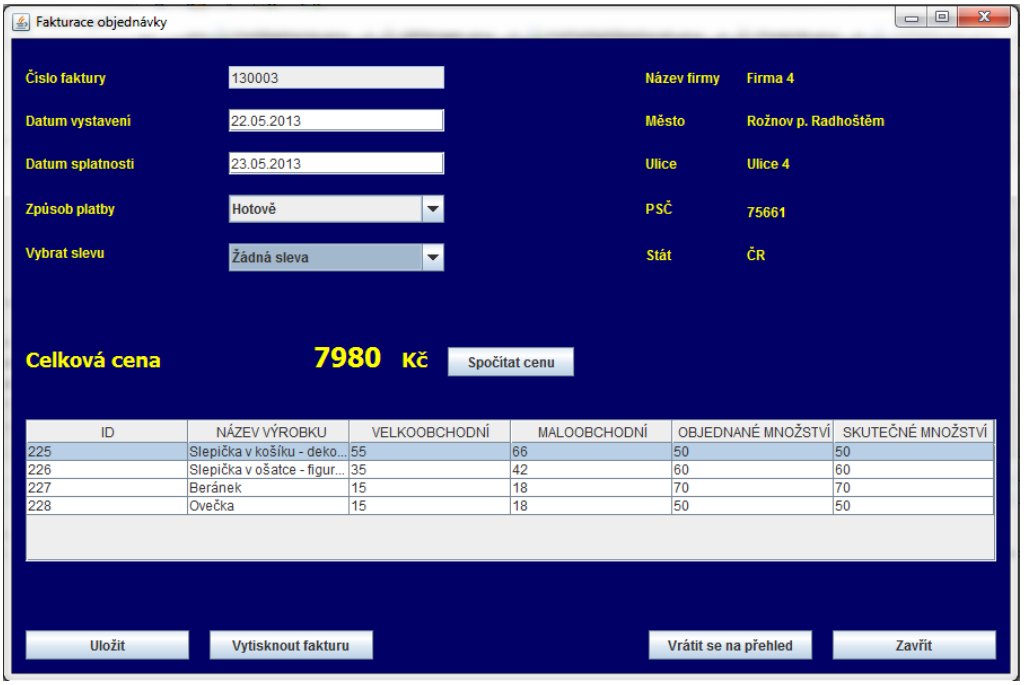
Díky tomuto formuláři je možno vyfakturovat příslušnou objednávku a fakturu později, pokud je třeba, editovat a exportovat do formátu xls. Exportu faktury je věnována další podkapitola.

Při inicializaci formuláře je třeba zjistit, zda je faktura už vystavena, či nikoliv. Podle toho se generuje, nebo načte id, které se vyplní do příslušného textového pole, jež není možno editovat, aby se zabránilo případným chybám při ukládání do databáze. Zobrazení buď nové, nebo už vytvořené faktury zajišťuje podmínka v tlačítku „Faktura“ ve formuláři FrmPrehledObjednavek. Pokud je záznam ve sloupci „Fakturováno“ tabulky tohoto formuláře prázdný, předá se formuláři FrmFakturace parametr 0, jinak pošle jako parametr id faktury, čímž se formulář zobrazí právě s touto fakturou, která se poté do něj načte. Při vytvoření nové faktury se automaticky generuje id, které má rozmezí 130001 – 139999. První vystavená faktura má hodnotu id 130001, kde se první dvojčíslí načítá z účetního roku, který je uložen v podformuláři frmNastaveni, pomocí metody getUcetniRok() a další 4 čísla jsou přidána pomocí řetězce „0001“.

Poté se takto vytvořený stringový řetězec převede na integer pro snadnější operace s id. Další záznam má proto logicky id 130002, kdy se pomocí příkazu Select prochází tabulka s fakturami a hledá se maximální hodnota id. Pokud se najde, navýší se o jedničku a může sloužit jako id nově založené faktury.

Objednané výrobky se načtou do tabulky tblPolozkyObjednavkyFakturace podle hodnoty boolean s názvem lNova. Pokud se lNova rovná false, což znamená, že faktura se pouze edituje - je už vytvořena, načtou se objednané výrobky do tabulky, do hlavičky se načte z resultSetu datum vystavení, datum splatnosti, způsob platby a do labelu celková cena. Jinými slovy se vybere už existující záznam a tím se formulář naplní. Jestliže je lNova true, naplní se pouze textové pole s id faktury (jeho generování je popsáno výše) a tabulka tblPolozkyObjednavkyFakturace s tím rozdílem, že není zadané ještě skutečně vyrobené množství, jež může později uživatel editovat, ale je implicitně předáno pomocí cyklu for ze sloupce objednané množství. Kód celého procesu inicializace formuláře je obsažen v příloze č. 4.

K dalším komponentám formuláře (obrázek 4.17) patří tlačítko „Uložit“, které podle výše zmíněné proměnné lNova rozhodne, zda se jedná pouze o Update již stávající faktury, nebo se jedná o fakturu novou a je tedy třeba provést SQL příkaz Insert. Nehledě na hodnotu lNova je proveden Update tabulky a atributu skutečně vyrobeného množství.



4.17 Formulář fakturace

Tlačítko „Vrátit se na přehled“ zavře formulář s fakturou a otevře znovu přehled objednávek. Formulář se ovšem po stisku tohoto tlačítka neuloží. Obdobně funguje tlačítko zavřít s tím rozdílem, že formulář pouze zavře.

Pro konečnou fakturaci je nejdůležitější samozřejmě konečná cena. Ta se počítá pomocí metody celkovaCena(), která je volána tlačítkem „Spočítat cenu“. V metodě je hlavní částí cyklus for, který postupně prochází celou tabulku, násobí mezi sebou na jednotlivých řádcích maloobchodní cenu a skutečné množství a tyto hodnoty postupně sčítá a ukládá do proměnné, která se poté převodem na stringový řetězec zobrazí v labelu. Cena ještě může být upravena pomocí komponenty JCombobox se slevou, která je načítána obdobně jako firmy ve formuláři s novou objednávkou a podle id odečte příslušné procento s celkové slevy. Velikosti a typy slev lze nastavit v číselníku.

V pravém horním rohu obrazovky jsou zobrazeny údaje o firmě, pro kterou je faktura vystavena, které slouží k tisku faktury. Jedná se pouze o labely, do kterých jsou pomocí resultSetu načtena data z tabulek *tbl_objednavka* a *tbl_zakaznik*.

4.3.2.6 *Export faktury do formátu xls*

Export faktury do formátu xls je zajištěn pomocí metody fakturaExport(). Principem této metody je vytvořit kopii šablony faktury (ta byla stažena pomocí programu Microsoft Excel z Microsoft Office Online) a do ní, respektive do její instance načíst data z formuláře. Tisk faktury by nebyl možný bez importu knihovny jxl 2.6.12.jar, která obsahuje třídy a metody, díky kterým je možno fakturu modifikovat podle vlastních potřeb. Mezi nejvýznamnější třídy patří například WritableCellFormat, nebo WritableFont. Jak už název tříd napovídá, díky nim lze nastavit pomocí parametrů formát buněk, potažmo fontu v buňkách.

Hlavička faktury, kde se nachází informace o firmě Medové Pečivo a zákazníkovi se do excelu načte jednoduše pomocí třídy Label a metody getWritableCell(), která vytvoří instanci buňky připravenou pro plnění daty. Tento krok je nutno opakovat pro každou jednu buňku, do které je třeba vkládat data. Tabulka s výrobky se plní pomocí cyklu for, ve kterém se načítají postupně název, množství, cena a cena objednaného množství jednotlivého výrobku. Zápis do excelu probíhá pomocí tříd Label (pro text), Number (pro import číselných údajů) a metody addCell(), která na rozdíl od getWritableCell() přidá buňku v excelu a poté vytvoří instanci pro naplnění daty. Tisk celkové ceny se provádí stejně jako v případě tisku cen do tabulky, nicméně už mimo cyklus for. Celý proces exportu hodnot je obalen do bloku try/catch, kdy v případě chyby se objeví okno s chybovým hlášením.

Výstup ve formátu xls je demonstrován v příloze č. 5.

4.3.2.7 *Implementace aplikace*

Protože aplikace byla vyvíjena na témže počítači, na kterém se s ní bude pracovat není potřeba žádné další instalace vyjma databázového systému Oracle, který je stejně nezbytný pro vývoj, tudíž musel být nainstalován jako první. Pro lepší přístup k souboru jar, kterým se aplikace spouští, je vytvořen zástupce na ploše s adresou tohoto souboru. Nyní už nebrání nic používání aplikace.

Za zmínku také stojí, že databáze obsahuje vyjma číselníku s výrobky pouze zkušební data, jelikož se jedná pouze o návrh. Uživatel si už firmy, slevy a objednávky samozřejmě musí zadávat sám.

5 Závěr

V úvodu této bakalářské práce byl stanoven cíl vytvořit návrh a realizovat databázovou aplikaci, která vede záznamy o objednávkách, fakturách, zákaznících a výrobcích. Na základě definování těchto cílů a požadavků firmy lze tvrdit, že se všechny tyto cíle povedlo beze zbytku splnit.

V druhé kapitole byly uvedeny všechny důležité teoretické základy, které byly potřebné k realizaci samotného projektu. Šlo zejména o kvalitní navržení modelu databáze pomocí tříúrovňového modelování, aby byl co nejpřesnější a nejefektivnější pro manipulaci s daty. Další část této kapitoly byla zaměřena na popis základních vlastností programovacího jazyka Java, které bylo třeba použít pro vývoj aplikace.

Samotný popis vývoje aplikace od vytvoření databáze pomocí SQL Developeru až po realizaci oken v Javě, je nastíněn v kapitole číslo čtyři. V této kapitole byla popsána jednotlivá okna jak z uživatelského, tak programového hlediska, kde byly vybrány z kódu ty nejdůležitější metody a třídy obsluhující běh aplikace, které lze najít v přílohách. Samotná aplikace se skládá ze tří částí. První jsou číselníky sloužící k editaci dat, druhá zadávání objednávky a třetí je výsledná fakturace a vytvoření výstupu programu v podobě faktury. Celá aplikace byla navržena tak, aby byla co nejjednodušší, ale zároveň efektivně splňovala všechny zadané požadavky.

Tím pádem je aplikace plně připravena pro běžné užívání, ovšem zdaleka nelze říct, že by byl vývoj ukončen a nebylo třeba provádět žádné změny. Některé nedostatky se jistě mohou objevit, stejně jako potřeba aktualizovat nebo vyladit některé stávající funkce, aby bylo ovládání ještě jednodušší, nicméně v současné podobě je aplikace funkční a dá se říct, že má i přínos v reálném světě.

Seznam použité literatury

Literatura:

[1] CHAPMAN, Stephen J. Začínáme programovat v jazyce JAVA. Praha: Computer Press, 2001. ISBN 80-7226-472-9.

[2] KALUŽA, Jindřich a Ludmila KALUŽOVÁ. *Modelování dat v informačních systémech*. I. vydání. Praha: Ekopress, 2012. ISBN 978-80-86929-81-1.

[3] LACKO, Luboslav. *SQL hotová řešení*. Brno: Computer Press, 2003. ISBN 80-7226-975-5.

[4] LONEY, Kevin a Marlene THERIAULT. *Mistrovství v Oracle*. Praha: Computer Press, 2002. ISBN 80-7226-635-7

[5] PECINOVSKÝ, Rudolf. Java 5.0: Novinky jazyka a upgrade aplikací. Brno: CP Books, 2005. ISBN 80-251-0615-2.

[6] POKORNÝ, Jaroslav a Ivan HALAŠKA. *Databázové systémy*. 2. přepracované vydání. Praha: ČVUT, 2003. ISBN 80-01-02789-9.

[7] ŠIMONOVÁ, Stanislava a Jan PANUŠ. *Databázové systémy I*. Pardubice: Univerzita Pardubice, 2007. ISBN 978-80-7194-988-6-55-764-07

[8] URMAN, S., R. HARDMAN a M. MCLAUGHLIN. *Oracle: Programování v PL/SQL*. Brno: Computer Press, 2007. ISBN 978-80-251-1870-2.

Online zdroje:

[1i] NetBeans IDE Features. NetBeans IDE [online]. 2011 [cit. 2013-03-22]. Dostupné z: http://netbeans.org/features/index.html

[2i] ORACLE. Oracle [online]. 2013 [cit. 2013-02-19]. Dostupné z: http://www.oracle.com/us/products/index.html

[3i] PANSKÝ, Mikoláš. Jedenáctý Oracle poprvé. Databázový svět: informační portál ze světa databázových technologií [online]. 2007 [cit. 2013-02-19]. Dostupné z: http://www.dbsvet.cz/view.php?cisloclanku=2007091401

[4i] ŠEDA, Jan. Úvod do JDBC. *Interval* [online]. 2003 [cit. 2013-02-25]. Dostupné z: http://interval.cz/clanky/uvod-do-jdbc/

[5i] UKLÁDÁNÍ A EDITACE DAT: Hirarchická databáze. Kartografie a Geoinformatika: Multimediální učebnice [online]. [cit. 2013-04-25]. Dostupné z: http://oldgeogr.muni.cz/ucebnice/kartografie/obsah.php?show=33&&jazyk=cz

[6i] VŠE [online]. 2005 [cit. 2013-02-25]. Dostupné z: http://java.vse.cz/pdf/Skripta383-databaze.pdf

Seznam zkratek

1NF	První normální forma
2NF	Druhá normální forma
3NF	Třetí normální forma
API	Application Programming Interfaces
BCNF	Boyce Coddova normální forma
CIA	Central Intelligence Agency
COBOL	Common Business Oriented Language
CODASYL	Conference on Data Systems Languages
CSV	Comma Separated Values
DBMS	Database Management System
DBTG	Database Task Group
DDL	Data Definition Language
DML	Data Manipulation Language
EE	Enterprise Edition
HTML	Hyper Text Markup Language
IBM	International Business Machines Corporation
IČ	Identifikační číslo
IDE	Integrated Development Environment
IMS	Information Management System
JDBC	Java Database Connectivity
JSP	Java Server Pages
JVM	Java Virtual Machine

ME	Micro Edition
OCI	Oracle Call Interface
ODBC	Open Database Connectivity
PHP	Hypertext Preprocessor
PL/SQL	Procedural Language/Structured Query Language
PSČ	Poštovní směrovací číslo
SE	Standard Edition
SPARC	Scalable Processor Architecture
SQL	Structured Query Language
UML	Unified Modeling Language
XMLA	Extensible Markup Language for Analysis

Prohlášení o využití výsledků bakalářské práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, bakalářskou práci užít (§ 35 odst. 3);
- souhlasím s tím, že bakalářská práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího bakalářské práce. Souhlasím s tím, že bibliografické údaje o bakalářské práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, bakalářskou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 10. května 2013

Václav Homola

.....

Václav Homola

Seznam příloh

Příloha č. 1: Metody rawSql() a rawSqlQuery()

```
public int rawSql(String cSql) throws SQLException {
        int lOk = -1;
        Statement stat = null;
        try {
            stat = con.createStatement();
            stat.executeUpdate(cSql);
            lOk = 1;

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (stat != null) {
                    stat.close();
                }
            } catch (SQLException e) {
            }
            return lOk;
        }

    } //provádí sql příkaz insert, update, nebo delete


public ResultSet rawSqlQuery(String iSql) {
        Statement stat = null;
        ResultSet rs = null;
        try {
            stat = con.createStatement();
            rs = stat.executeQuery(iSql);

        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
        return rs;


    } //provede select
```

Příloha č.2: metoda getMarze()

```
public float getMarze(String iSql) {
        float marze = 0.0F;
        Statement stat = null;
        ResultSet rs = null;
        try {
            stat = con.createStatement();
            rs = stat.executeQuery(iSql);
            if (rs.next()) {
                System.out.println(rs.getString("marze"));
                marze = rs.getFloat("marze");
            }
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stat != null) {
                    stat.close();
                }

            } catch (SQLException e) {
            }
            return marze;
        }
```

Příloha č. 3: metoda fillData()

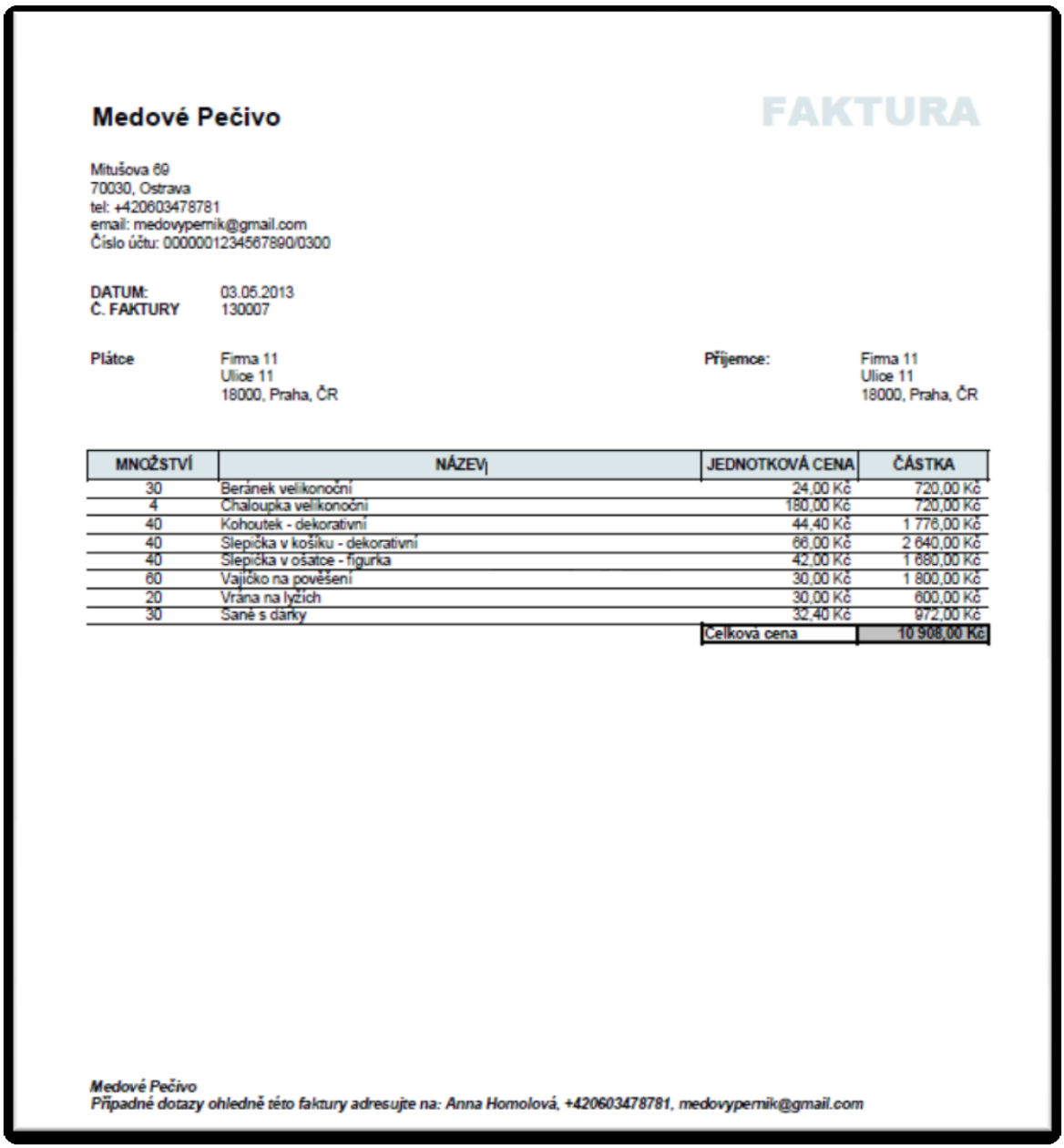
```
private void fillData() {
        //obnovuje tabulku s ResultSetem po delete a update
        String cSql = "SELECT  P.id_polozky_objednavky AS ID, V.nazev AS
Název, "
                + "P.mnozstvi_objednane AS Množství, P.velko_cena AS
Velkoobchodní, P.malo_cena AS Maloobchodní, P.poznamka AS Poznámka "
                + "FROM bakalar.tbl_polozky_objednavky P "
                + "INNER JOIN bakalar.tbl_vyrobky V ON P.id_vyrobku =
V.id_vyrobku "
                + "WHERE id_objednavky= " + idObj;
        ResultSet rs = dbM.rawSqlQuery(cSql);

        TableModel model2 = DbUtils.resultSetToTableModel(rs);
        tblPolozkyObjednavky.setModel(model2);

        try {
            if (rs != null) {
                rs.close();
            }

        } catch (SQLException ex) {
            Logger.getLogger(FrmObjednavka.class
                    .getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Příloha č. 4: Inicializace formuláře FrmFakturace

```
public FrmFakturace(int pIdFak, int pIdObj) throws SQLException {
        initComponents();
        dbM.connectDb();
        idObj = pIdObj;
        txtCisloFaktury.setEditable(false);

        if (pIdFak == 0) {
            btnTiskFaktury.setEnabled(false);
            lNova = true;
            String ucetniRok = dbM.getUcetniRok("SELECT ucetni_rok FROM
bakalar.app_setup");
            int idFakOd = Integer.valueOf(ucetniRok.substring(2, 4) +
"0001");
            int idFakDo = Integer.valueOf(ucetniRok.substring(2, 4) +
"9999");
            cSql = "SELECT MAX(id_faktury) AS id_faktury FROM
bakalar.tbl_fakturace "
                    + "WHERE id_faktury BETWEEN " + idFakOd + " AND " +
idFakDo;
            ResultSet rs = dbM.rawSqlQuery(cSql);
            rs.next();
            int id = rs.getInt("id_faktury");
            if (id == 0) {
                idFak = Integer.valueOf(ucetniRok.substring(2, 4) + "0001");
            } else {
                idFak = ++id;
                try {
                    if (rs != null) {
                        rs.close();
                    }
                } catch (SQLException e) {
                    System.out.println(e);
                }
            }

        } else {
            lNova = false;
            idFak = pIdFak;
            cSql = "SELECT datum_splatnosti, "
                    + "datum_vystaveni, zpusob_platby, celkova_cena "
                    + "FROM bakalar.tbl_fakturace "
                    + "WHERE id_objednavky = " + idObj;
            ResultSet rs = dbM.rawSqlQuery(cSql);
            rs.next();

txtDatumVystaveni.setText(mc.getDateFormat(rs.getString("datum_vystaveni")));
txtDatumSplatnosti.setText(mc.getDateFormat(rs.getString("datum_splatnosti"))
);
            cmbZpusobPlatby.setSelectedItem(rs.getString("zpusob_platby"));
            lblCelkovaCena.setText(rs.getString("celkova_cena"));
            try {
                if (rs != null) {
                    rs.close();
                }
            } catch (SQLException e) {
                System.out.println(e);
            }
```

```
        }
        cSql = "SELECT O.id_polozky_objednavky AS ID, V.nazev AS \" NÁZEV
VÝROBKU\", O.velko_cena AS \"VELKOOBCHODNÍ\", "
                + "O.malo_cena AS\"MALOOBCHODNÍ\", "
                + "O.mnozstvi_objednane AS \"OBJEDNANÉ MNOŽSTVÍ\",
O.mnozstvi_skutecne AS \"SKUTEČNÉ MNOŽSTVÍ\" "
                + "FROM bakalar.tbl_polozky_objednavky O INNER JOIN
bakalar.tbl_vyrobky V "
                + "ON O.id_vyrobku=V.id_vyrobku "
                + "WHERE O.id_objednavky = " + pIdObj;
        ResultSet rs = dbM.rawSqlQuery(cSql);
        TableModel model = DbUtils.resultSetToTableModel(rs);
        tblPolozkyObjednavkyFakturace.setModel(model);
        tblPolozkyObjednavkyFakturace.changeSelection(0, 0, false, false);
        txtCisloFaktury.setText(idFak.toString());
        if (lNova == true) {
            for (int i = 0; i < tblPolozkyObjednavkyFakturace.getRowCount();
i++) {
                BigDecimal bd =
mc.getBigDecimal(tblPolozkyObjednavkyFakturace.getValueAt(i, 4));
                int mnozstvi = bd.intValue();
                tblPolozkyObjednavkyFakturace.setValueAt(mnozstvi, i, 5);
                /*objednané množství je implicitně předáno do skutečného, ze
kterého se později počítá celková cena v metodě celkovaCena(). lze jej
editovat... */
            }
        }
        try {
            if (rs != null) {
                rs.close();
            }
        } catch (SQLException e) {
            System.out.println(e);
        }

        addFirma();

    }
```

Příloha č. 5: Faktura ve formátu xls



Medové Pečivo

FAKTURA

Mitušova 69
70030, Ostrava
tel: +420603478781
email: medovypernik@gmail.com
Číslo účtu: 0000001234567890/0300

DATUM: 03.05.2013
Č. FAKTURY 130007

Plátce
Firma 11
Ulice 11
18000, Praha, ČR

Příjemce:
Firma 11
Ulice 11
18000, Praha, ČR

MNOŽSTVÍ	NÁZEV	JEDNOTKOVÁ CENA	ČÁSTKA
30	Beránek velikonoční	24,00 Kč	720,00 Kč
4	Chaloupka velikonoční	180,00 Kč	720,00 Kč
40	Kohoutek - dekorativní	44,40 Kč	1 776,00 Kč
40	Slepička v košíku - dekorativní	66,00 Kč	2 640,00 Kč
40	Slepička v ošatce - figurka	42,00 Kč	1 680,00 Kč
60	Vajíčko na pověšení	30,00 Kč	1 800,00 Kč
20	Vrána na lyžích	30,00 Kč	600,00 Kč
30	Saně s dárky	32,40 Kč	972,00 Kč
		Celková cena	10 908,00 Kč

Medové Pečivo
Případné dotazy ohledně této faktury adresujte na: Anna Homolová, +420603478781, medovypernik@gmail.com